Theory and Practice of Natural Strategy Checking in Concurrent Games with Imperfect Information

Marco Aruta[1], Francesco Improta[1], Vadim Malvone[2], and Aniello Murano[1]

[1] Università degli Studi di Napoli Federico II, Naples, Italy
[2] LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France

Abstract. In this paper, we explore strategy-checking algorithms in concurrent games with imperfect information, analyzing their computational complexity in comparison to the perfect information setting. Precisely, we analyze *uniform natural strategies* used by agents with limited memory and computational capacity. We examine the decision problems of *Sure Winning* and *IsNash*, as well as other similar decision problems in formal strategic reasoning. Additionally, we develop a tool that demonstrates the practical applicability of the proposed framework.

Keywords: Multi-Agent Systems, Natural Strategies, Concurrent Games

1 INTRODUCTION

Formal strategic reasoning plays a crucial role in addressing decision-making challenges in artificial intelligence. It offers a rigorous foundation for designing systems where autonomous agents interact and make choices under uncertainty. In practical scenarios, agents operate with imperfect information due to limited observability. Concurrent game structures with imperfect information (iCGS) [10] provide a formal framework that models these scenarios using indistinguishability relations to capture the partial view of each agent of the state space. Moreover, temporal and strategic logics are essential to specify and verify the evolution of system behaviors over time, ensuring that strategic decisions are rigorously assessed within complex and unpredictable operational contexts [7]. Recent work on natural strategies [8,9] has introduced a simplified rule-based approach to strategic reasoning, where condition-action rules encode an agent's decision process in line with bounded rationality. Such natural strategies are particularly appealing for agents with limited computational resources because they emphasize ease of execution and understandability. However, as highlighted in [8], existing frameworks for natural strategic ability have been developed under the assumption of perfect information. This focus limits their applicability since, in real-world settings, agents must operate under imperfect information. This motivates our work.

Our Contribution. In this paper, we extend the state-of-the-art in strategic reasoning for concurrent games with imperfect information by integrating natural strategies with explicit Linear Temporal Logic (LTL) objectives. First, we study for the first time the decision problems of *Sure Winning*, *IsNash*, as well as other similar decision problems in formal strategic reasoning, tailored to natural strategies in the imperfect information setting, where LTL objectives are employed to rigorously specify agents' temporal

goals. Second, leveraging the obtained theoretical results, we develop a novel tool and, as a practical demonstration, we apply it to a specific istantiation of the well-known gossip protocols-drawing [6]; precisely we convert this protocol into an iCGS model by mapping service and gossip actions accordingly. This example allows us to illustrate how our method can effectively be used to model and verify agents' behaviors in distributed systems under imperfect information through natural strategies. Our approach suits scenarios like swarm robotics and sensor networks (e.g., drones in disaster zones), where agents act on local observations using simple condition-action rules and LTL goals or industrial automation (e.g., robotic arms), where behaviors are driven by current sensor inputs, with LTL objectives.

Related work. The study of strategic ability under both *perfect information* and *imperfect information* has been dominated by the development of different temporal logic for strategic reasoning. More recently, in [8,9] has been introduced the notion of *natural strategies* in perfect information setting. Extensions to *imperfect information*, where epistemic indistinguishability is added to capture agents' limited observations [10], lacks of work combining natural strategies with LTL objectives under imperfect information [5,4]. In classical game theory one distinguishes *pure strategies* - which deterministically assign an action to each history of play - from *mixed* (non-pure) strategies, where a player randomizes over pure strategies according to a probability distribution [12]. In this work, we focus on uniform pure natural strategies of bounded complexity under imperfect information.

2 BACKGROUND

We briefly supply the theoretical tools we use throughout. First, we introduce LTL, a modal logic used to describe sequences of states in a linear progression over time. It extends propositional logic by introducing temporal operators that allow reasoning about the timing of events. LTL is useful in computer science, especially in formal verification [13]. Let $Prop$ be the set of atomic propositions with the typical element p. The syntax of LTL is formally defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathrm{X}\varphi \mid \varphi \mathrm{U} \varphi$$

Additional propositional operators ($\vee$, $\rightarrow$ etc.) are defined as usual. Furthermore, we define temporal operators $\mathrm{F}\varphi \equiv \top \mathrm{U}\varphi$ ("sometime in the future"); $\mathrm{G}\varphi \equiv \neg\mathrm{F}\neg\varphi$ ("always from now on"); $\varphi_1 \mathrm{W} \varphi_2 \equiv \varphi_1 \mathrm{U} \varphi_2 \vee \mathrm{G}\varphi_1$ ("weak until").

LTL is semantically defined through behavioural models, namely limitless paths derived from the execution of a Labeling Transition System (LTS). Let λ be a path in an LTS, as for notation we set: (*i*) $\lambda[i]$ is the *i*th position on λ (starting at 0); (*ii*) $\lambda[i \dots j]$ is the part of λ amid positions i and j; (*iii*) $\lambda[i \dots \infty]$ is the suffix of λ starting at position i.

With I being the interpretation function in an LTS, the semantics follow:

- $\lambda \models p \iff \lambda[0] \in I(p)$;
- $\lambda \models \neg\varphi \iff \lambda \not\models \varphi$;

- $\lambda \models \varphi_1 \wedge \varphi_2 \iff \lambda \models \varphi_1$ and $\lambda \models \varphi_2$;
- $\lambda \models X\varphi \iff \lambda[1 \ldots \infty] \models \varphi$;
- $\lambda \models \varphi_1 U \varphi_2 \iff \lambda[i \ldots \infty] \models \varphi_2$ for some $i \geq 0$ and $\lambda[j \ldots \infty] \models \varphi_1$ for all $0 \leq j < i$.

We present an *iCGS* as a tuple $iM = \langle Agt, St, Act, d, t, Prop, V, \{\sim_i\}_{i \in Agt} \rangle$ where:

- Agt is the non empty finite set of agents $Agt = \{a_1, ...a_{|Agt|}\}$;
- St is the non empty finite set of states;
- Act is the non empty finite set of actions;
- $Prop$ is the non empty finite set of atomic propositions;
- $V : St \to 2^{Prop}$ is the propositional valuation;
- $d : Agt \times St \to 2^{Act}$ is a function that defines availability of action in a state;
- t is the deterministic transition function that assigns a successor state, denoted as $q' = t(q, \alpha_1, \ldots, \alpha_{|\mathrm{Agt}|})$, to each state $q \in \mathrm{St}$ and any tuple of actions $\alpha_i \in d_a(q)$ that can be executed by Agt in q;
- for every $a_i \in \mathrm{Agt}$, $\sim_i$ is a relation of indistinguishability between states, that is, given two states $s, s' \in \mathrm{St}$, $s \sim_i s'$ *iff* s and s' are indistinguishable for agent a_i.

Furthermore, the *iCGS* is uniform, which means that $q \sim_i q' \to d_i(q) = d_i(q')$. Let the tuple $iG = \langle iM, q_0, \phi \rangle$ be a concurrent game with imperfect information and binary winning condition, where iM is an iCGS, $q_0 \in St$ is a state of iM, and $\phi : Agt \to \mathcal{L}_{LTL}$ is a mapping that assigns an LTL formula to each agent, representing desired outcomes. By $\Lambda^{iM} \subseteq \mathrm{St}^{\omega}$, we denote the set of all the paths in iM and by $\Lambda^{iM}(q)$ all the paths in iM starting in q. Similarly, a history $h = q_0 q_1 q_2 \ldots q_n$ is a finite sequence of states that can be effected by subsequent transitions. By $\mathrm{last}(h) = q_n$ we denote the last element of the sequence. Finally, $H^{iM} \subseteq \mathrm{St}^{+}$ is the set of all the histories in iM.

Here, natural strategies consist of lists of condition-action rules, where epistemic conditions are expressed as Boolean propositional formulas. The first rule whose condition holds in the current state is selected, and the corresponding action is executed. Formally, let $\mathcal{B}(\Xi)$ be the set of Boolean formulas over alphabet Ξ. We represent natural strategies by lists of guarded actions, i.e., sequences of pairs (ϕ_i, α_i) such that:

- $\phi_i \in \mathcal{B}(AP)$;
- $\alpha_i \in d_a(q)$ for every $q \in$ such that $q \models \phi_i$.

That is, ϕ_i is a propositional condition on states of the iCGS, and α_i is an action available to agent a in every state where ϕ_i holds. We define natural strategy complexity as the number of propositions along the conditions of the natural strategy. For a detailed description of natural strategy complexities, see [8].

3 DECISION PROBLEMS UNDER IMPERFECT INFORMATION

The main contribution of this work is to extend key decision problems in perfect information games studied under “Natural Strategic Abilities" [8] to the imperfect information setting, by means of and indistinguishability relation among states. Natural strategies rely on condition-action rules that apply consistently across all indistinguishable

states. As previously stated, in iCGS we emphasize that agents operate under strategies constrained by limited visibility, dictated by an indistinguishability relation. Therefore, we need an epistemic operator K in our conditions. Formally, we define the set of epistemic conditions for an agent a as follows:

$$\psi ::= \top \mid K_a\varphi \mid \neg\psi \mid \psi \wedge \psi$$
$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_b\varphi$$

where p is an atomic proposition and, a and b are agents. Given an *iCGS iM*, a state $q \in St$, and an epistemic condition φ, we inductively define the semantics:

- $q \models p \iff p \in V(q)$;
- $q \models \neg\varphi \iff q \not\models \varphi$;
- $q \models \varphi_1 \wedge \varphi_2 \iff q \models \varphi_1$ and $q \models \varphi_2$;
- $q \models K_a\varphi \iff$ for all $q' \sim_a q$, $q' \models \varphi$.

Note that natural strategies are inherently uniform as proved in Proposition 2.3 in [10], namely they specify the same actions in indistinguishable states. The following subsections describe Surely Winning and Nash Equilibrium decision problems in *iCGS*.

3.1 Surely Winning

We introduce the most popular solution concept in logical approaches to strategic reasoning: surely winning. Agent a wins if he has a uniform strategy that obtains his goal no matter what the other agents do.

Algorithm 1 SUREWIN($\mathcal{G}, A, k$)

1: $s_A \leftarrow$ GUESSSTRAT($\mathcal{G}, A, k$) ▷ Guess a uniform strategy for coalition A within bound k
2: Prune model $\mathcal{M}$ according to s_A, obtaining $\mathcal{M}'$ ▷ Simplify model via guessed strategy
3: **return** MCHECK$_{\text{CTL}}(\mathcal{M}', q_0, \text{A} \bigwedge_{i \in A} \Phi_i)$ ▷ Verify coalition A always wins in pruned model

Definition 1. *Given a concurrent game with imperfect information $iG = \langle iM, q_0, \Phi \rangle$, a set of agents $A \in Agt$, and $k \in \mathbb{N}$, we say that s_A is surely winning in iG* iff *for all $\lambda \in out(q_0, s_A)$ and $a \in A$ it holds that $\lambda \models \Phi_a$. Also, coalition A surely wins in iG under bound k if it has a uniform sure winning strategy with a size no greater than k.*

Input: A game *iG*, a coalition A, and a natural number k.

Output: true if coalition A surely wins in *iG* under bound k; false otherwise.

Given the above definition, we provide the related complexity result.

Proposition 1. *SureWin is in PSPACE.*

Proof. As described in Algorithm 1, let G be a finite graph, and A be a coalition with a natural strategy. Since k is a numeric bound given in unary, the entire list of strategies is at most polynomial in the input size. We nondeterministically enumerate the at-most-k conditions. At any moment, we only maintain the current partial list and a

counter, which requires logarithmic space. Once a candidate strategy s_A is guessed, we force the coalition to agree with s_A. This generates a computation tree in polynomial space. To verify whether all paths of this tree satisfy the coalition's LTL objective, we invoke the standard automata-theoretic model checker for LTL, which runs in PSPACE with respect to the size of the graph. A guard like $K_i\phi$ does not increase the overall complexity. The entire process is nondeterministic-PSPACE, and by Savitch's theorem corollary, NPSPACE = PSPACE. Hence, SUREWIN is in PSPACE. □

3.2 Nash Equilibrium

Nash equilibrium is a key concept in game theory [12] that describes a situation in which each player's strategy is optimal, given the strategies chosen by others. In this equilibrium, no player can benefit by changing their strategy unilaterally; that is, if one player decides to alter their choice while others remain constant, he will not improve their outcome. This equilibrium can occur in pure strategies, where players select one specific action, or in mixed strategies, where they randomize their choices. Here, we focus on pure uniform strategies.

Definition 2 (Best response). *Given a concurrent game with imperfect information $iG = \langle iM, q_0, \Phi\rangle$, an agent i, and a profile $s_{Agt} = (s_1, \ldots, s_i, \ldots, s_{|Agt|})$ of uniform natural strategies under bound $k \in \mathbb{N}$, we say that s_i is a best response in s_{Agt} under bound k if and only if the path induced by s_{Agt} (that is, $path(s_{Agt})$) does not satisfy Φ_i implies that, for all $s'_i \in \Sigma_i^{nir}$ such that the complexity of s'_i is less or equal than k, also $path(s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_{|Agt|}) \not\models \Phi_i$.*

Definition 3 (Nash equilibrium). *A uniform natural strategy profile $s_{Agt} = (s_1, \ldots, s_{|Agt|})$ is a Nash Equilibrium in a concurrent game with imperfect information iG under bound k if and only if for every $i \in Agt$, s_i is a best response in s_{Agt} under bound k.*

Now, we present decision problems related to Nash equilibrium. The problems below come very close to the work of Wooldridge, Gutierrez and colleagues on equilibrium checking [15] [14] [2]. The main difference lies in our focus on uniform natural strategies bounded by k employed by agents, whereas their approach considers arbitrary combinatorial strategies with perfect recall.

IsNash We begin with the basic solution concept ISNASH, defined by specifying its required inputs and expected output.
Input: A game iG, a uniform strategy profile s_{Agt}, and a natural number k representing the bound.
Output: true if s_{Agt} is a Nash equilibrium in iG under bound k; false otherwise.

Proposition 2. *IsNash is in CoNP.*

Proof. In Algorithm 2, we show an algorithm that uses a NP procedure to check whether a uniform strategy profile s_{Agt} is not a Nash equilibrium. Starting with a strategy profile s_{Agt} we check that for each agent i, if the path generated by the uniform strategy

Algorithm 2 IsNotNash($\mathcal{G}, s_{\text{Agt}}, k$)

1: **for** each agent $i \in \text{Agt}$ **do** ▷ Iterate through all agents
2: **if** path$(s_{\text{Agt}}) \not\models \Phi_i$ **then**
3: Guess s'_i such that compl$(s'_i) \leq k$ ▷ Try deviation for agent i
4: Construct $s'_{\text{Agt}} = (s_1, \ldots, s_{i-1}, s'_i, s_{i+1}, \ldots, s_{|\text{Agt}|})$
5: **if** path$(s'_{\text{Agt}}) \models \Phi_i$ **then**
6: **return** true
7: **end if**
8: **end if**
9: **end for**
10: **return** false

profile does not satisfy his own goal Φ_i, then no unilaterally deviation of the agent i will allow him to satisfy his goal as well. Since CoNP is the class of problems whose complements are in NP, the result immediately follows. As previously stated proving Proposition 1, K being in the conditions employed in line 3 of the algorithm is not relevant in complexity terms. Then, we establish that IsNash is in CoNP in *iCGS*. □

ExistNash When considering only pure strategies (non-randomized strategies) there is no guarantee that a Nash equilibrium exists. We thus check whether the game has a stable point.

Input: A game *iG* and a natural number k.

Output: True if there exists a uniform strategy profile s_{Agt} that is a Nash equilibrium in *iG* under bound k; False otherwise.

Algorithm 3 ExistNash($\mathcal{G}, k$)

1: $s_{\text{Agt}} \leftarrow$ GuessStrat$(\mathcal{G}, \text{Agt}, k)$
2: **return** $\neg$ IsNotNash$(\mathcal{G}, s_{\text{Agt}}, k)$ ▷ Check if the guessed strategy is a Nash equilibrium

Proposition 3. *ExistNash is in* $\sum_2^P$.

Proof. We ask whether there exists a profile of natural strategies of size $\leq k$ such that no single player can profit by unilaterally switching to any other natural strategy of the same size. The proof is a two-layer search: we start by nondeterministically generating a full strategy list (again polynomial length thanks to the k bound); then, for each player i, we evaluate profitable deviations. It first checks the strategy correctness; if player i's LTL objective already holds, player i can perform. Otherwise, the oracle nondeterministically guesses one alternative natural strategy s'_i (size $\leq k$), re-simulates the game with s'_i in place of s_i, and checks - still in polynomial space - whether the resulting play now satisfies player i's objective. If such a profitable deviation exists, the oracle returns true (i.e., "is not a Nash equilibrium"). Guessing s'_i and running a single-path LTL check is plain NP. The outer algorithm is NP and calls an NP oracle once. This is exactly the class $\text{NP}^{\text{NP}} = \sum_2^P$ in the polynomial hierarchy. Any deviation s'_i must remain uniform;

the oracle checks this syntactically by comparing guards across indistinguishable states — again a polynomial test. Knowledge operators inside guards act like fresh propositional atoms, so they do not affect the complexity argument. Therefore, the whole search lies in $\sum_2^P$, proving that EXISTNASH $\in \sum_2^P$.□

WinsSomeNash We ask whether a given agent wins in some Nash equilibrium.
Input: A game *iG*, an agent *i*, and a natural number *k*.
Output: true if there exists a uniform strategy profile s_{Agt} that is a Nash equilibrium in *iG* under bound *k*, and $\text{path}(s_{\text{Agt}}) \models \Phi_i$; false otherwise.

Algorithm 4 WINSSOMENASH($\mathcal{G}, i, k$)

1: $s_{\text{Agt}} \leftarrow$ GUESSSTRAT($\mathcal{G}$, Agt, k)
2: **if** $\text{path}(s_{\text{Agt}}) \models \Phi_i$ **and** $\neg$ ISNOTNASH($\mathcal{G}, s_{\text{Agt}}, k$) **then**
3: **return** true
4: **end if**
5: **return** false

Proposition 4. *WinsSomeNash is in* $\sum_2^P$.

Proof. In Algorithm 4, we present an algorithm that employs an NP procedure to select the strategy profile s_{Agt}, calling the NP procedure described in Algorithm 2, which checks whether s_{Agt} is a Nash equilibrium. We highlight that given a path λ and an LTL formula φ, checking whether λ satisfies φ requires polynomial time [11]. Furthermore, the epistemic operator K does not increase complexity. Therefore, we prove that the total complexity to decide WinsSomeNash in *iCGS* is $\sum_2^P$. □

WinsAllNash We ask whether a given agent wins in all Nash equilibria.
Input: A game *iG*, an agent *i*, and a natural number *k*.
Output: true if for every uniform strategy profile s_{Agt} that is a Nash equilibrium in *iG* under bound *k*, $\text{path}(s_{\text{Agt}}) \models \Phi_i$; false otherwise.

Algorithm 5 LOSESSOMENASH($\mathcal{G}, i, k$)

1: $s_{\text{Agt}} \leftarrow$ GUESSSTRAT($\mathcal{G}$, Agt, k)
2: **if** $\neg(\text{path}(s_{\text{Agt}}) \models \Phi_i)$ **and** $\neg$ ISNOTNASH($\mathcal{G}, s_{\text{Agt}}, k$) **then**
3: **return true**
4: **end if**
5: **return false**

Proposition 5. *WinsAllNash is in* Π_2^P.

Proof. In Algorithm 5, we present LosesSomeNash, the algorithm to check the complement problem of WinsAllNash. The algorithm uses a NP procedure to select the

strategy profile s_{Agt} and calls another NP procedure (Algorithm 2) to check whether s_{Agt} is a Nash equilibrium. So, LosesSomeNash in in $\sum_2^P$ and thus WinsAllNash is in Π_2^P. Again, guessing strategy is not influenced by the presence of K. Therefore, we prove that WinsAllNash is in Π_2^P in *iCGS*. □

4 IMPLEMENTATION

In this section, we describe the practical realization of our tool for concurrent games with imperfect information. Our Python implementation embodies the theoretical concepts of uniform natural strategies and decision problems introduced earlier. The system is modular, supporting both automated strategy generation and manual strategy input, and leverages parallel processing to manage computationally intensive tasks within predefined time constraints.

4.1 Tool Architecture

Our implementation is structured around two principal modules that mirror the key components of our theoretical framework:

1. Natural Strategies Generation This module is responsible for the generation and refinement of natural strategies. The correctness of the natural strategies generated is certified by using the same strategy generation tool described in [1], which has been extended for the present use-case. It includes:

- `generate_guarded_action_pairs`: Computes Cartesian products between conditions (derived from atomic propositions) and the available actions for each agent.
- `generate_strategies`: Constructs complete strategy profiles just by combining guarded action pairs while ensuring that the overall complexity (measured by the number of tokens in conditions, plus an extra cost for negations) does not exceed the bound k.
- `generate_deviations_for_agent`: Explores potential unilateral deviations for each agent's strategy, guaranteeing that any alternative considered respects the complexity constraint.

2. Model Pruning This module implements the pruning function that reduces the model to capture the unique execution path induced by a given strategy - corresponding to the formal `path()` function defined in our work. The pruning process:

- Restricts each state $q \in St$ to only those actions consistent with the strategy, thereby eliminating transitions that do not conform.
- Updates the transition function t accordingly, ensuring that the resulting pruned model M' is polynomially bounded by the original model M.
- Facilitates verification by reducing the initial model. Our tool addresses a fragment of LTL that excludes nesting temporal operators. Given this limitation, we can reduce the verification to Computation Tree Logic (CTL) [3] model checking. CTL, in addition to temporal operators, introduces path quantifiers E ("there is a

path") and *A* ("for every path"). Thus, we prefix a universal path quantifier to the LTL formula under exam. This approach simplifies the verification process and is particularly appropriate for this first version of our tool, where no similar method existed. Future extensions are planned to incorporate automata-based techniques to handle more complex LTL formulas.

Furthermore, our implementation supports both multi-agent and single-agent scenarios: (*i*) **Multi-agent case**: The `process_data` function orchestrates initialization, strategy generation, deviation exploration, and Nash equilibrium verification. Outcomes (including the complexity bound used, the winning strategy profile if found, and the execution time) are recorded in a designated result file; (*ii*) **Single-agent case**: The `sureWin` function manages strategy checking for scenarios involving a single agent, following a similar workflow tailored to the simpler decision problem.

4.2 Dual Strategy Generation

The framework supports two approaches for strategy creation.

Automated Strategy Generation. In the absence of user-specified strategies, the system automatically generates natural strategies by combining condition-action pairs. For each complexity bound from 1 to k, the system generates all valid guarded action pairs and combines them into complete strategy profiles. This process is guided by the requirement that the total complexity of the strategy must exactly match the bound k.

Manual Strategy Input. Users can also manually input natural strategies when required as inputs. The system prompts the user to enter a condition and an action. The condition is validated against a list of atomic propositions and checked to ensure its complexity does not exceed k, while the action is validated against the available actions for the agent. The entered pairs are stored in the same structured format as the automatically generated strategies, ensuring consistency in subsequent processing.

4.3 Experiments

We conducted extensive tests on the proposed verification tool using an HP Omen 15-ax213ng equipped with an Intel i7-7700HQ 3.8 GHz CPU and 16 GB of RAM. The development was carried out in Python 3.9 using PyCharm, with GitHub serving as the hosting service [3]. To validate the correctness of our tool, we performed more than 1000 tests. To initiate a preliminary experiment, it has been crucial to examine the correspondence between the Boolean solutions generated by our algorithm, compared to a canonic NatATL algorithm already implemented and validated that follows the same behavior, translating its formulas into CTL formula equivalents [8]. Since, in practice, both algorithm's behavior can vary depending on the input model and/or formula due to a complexity bound on the generated strategies, we noticed a perfect alignment into verification responses, for a total of 1000 tests. Each temporal operator was tested a

[3] The Github link to our tool is provided here: https://github.com/MarcoAruta/NatLTL

hundred times to understand its behavior (for a total of 400 tests). Identical inputs were used for both algorithms, with their corresponding formula versions. For computational purposes, our algorithm complexity bound was increased up to 5. In a comprehensive analysis of more than a thousand tests, it was conclusively demonstrated that our verification tool operates correctly.

Successively, we also assessed the time performance of our tool. In particular, when strategies are provided manually by the user (as `isNotNash()` does), responses are generated at runtime, resulting in prompt verification. In contrast, when invoking functions like `existsNash(), LoseSomeNash(), WinsSomeNash, sureWin()` functions - due to their reliance on automatic strategy generation - the execution time is much higher.

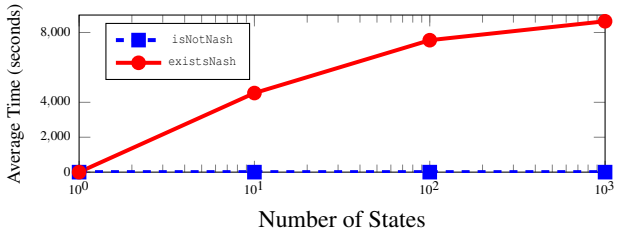

Fig. 1: Objective functions time comparison

P	S	sureWin()	isNotNash()	existsNash()	winsSomeNash()	loseSomeNash()
2	4	1764.68	0.02	1764.70	1764.63	1764.65
3	10	5531.19	0.02	5531.26	5531.33	5531.02
4	100	TO	0.03	TO	TO	TO
5	1000	TO	0.03	TO	TO	TO

Table 1: Algorithms execution times comparison with a fixed $k = 2$

Table 1 compares the average execution times of our functions that vary with the number of agents and states, with the complexity bound fixed at $k = 2$. This bound is chosen because automatically generating all possible natural strategies for $k > 2$ is prohibitively expensive due to their intrinsic complexity. As shown in Figure 1, `isNotNash` (blue line) demonstrates exceptional efficiency, executing in approximately 0.02 seconds even as the number of states increases. This remarkable speed highlights the robustness and scalability of `isNotNash`, especially when contrasted with the significantly longer runtimes of strategy-generation functions. These functions experience exponential growth in execution time, rising from 1764 seconds for 4 states and a single couple of agents to over 7200 seconds (our timeout threshold) for 1000 states and 5 agents. This stark contrast underscores the substantial computational burden of exploring the full space of potential natural strategies, further emphasizing the strategic advantage of the streamlined `isNotNash` approach.

4.4 Case Study: Gossip Protocol

We emphasize that the purpose of this section is to present a well-known case study from the literature, which we have integrated into our tool to provide experimental re-

sults. Our goal is not to directly compare our tool with that presented in [6] because (*i*) our approach develops natural strategies while the cited work employs classical ones, and (*ii*) the underlying solution concepts differ, precluding a fair comparison. In this case study, we illustrate our framework by converting the Simple Reactive Modules Language (SRML) - as used in the equilibrium verification approach of [6] - into an iCGS. Notably, the behavior of an SRML module naturally generates a Kripke structure, which forms the foundation of our game model, and the referenced tool operates on concurrent game structure graphs. Furthermore, the update rules in SRML directly translate into the set of actions available at each iCGS state, and LTL is employed to specify goals in both approaches. We illustrate our approach by converting the gossip protocols example from [6].

Gossip protocols, modeled after information spread in social networks, are key to ensuring data consistency in large-scale distributed systems through regular message exchanges. In our model, the service action (i.e., receiving information) is mapped to the `stay` action, while the gossip action remains unchanged. An agent involved in the protocol operate in two modes: *Servicing (S)* (when an agent is providing service, i.e., $s_i = \text{true}$) and *Gossiping (G)* (when $s_i = \text{false}$). Each agent aims to gossip infinitely often, as captured by the LTL formula $GF\,\neg s_i$. For a system with two agents, the state space consists of four states. In the (S,S) state, both agents independently choose to either remain servicing (`stay`) or switch to gossiping (`switch`). In the asymmetric states (S,G) and (G,S), the servicing agent chooses its action while the gossiping agent follows a fixed (default) behavior. Finally, in the (G,G) state - where neither agent satisfies an atomic proposition - a forced transition $(\text{forced}, \text{forced})$ returns the system to (S,S) to restore service. This iCGS caputures the dynamics of gossip protocols.

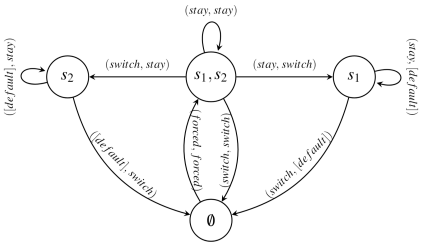

Fig. 2: Gossip Protocol iCGS

5 CONCLUSIONS

Our work extends natural strategy checking to concurrent games with imperfect information by integrating uniform natural strategies with epistemic reasoning. We rigorously proved that key decision problems maintain their complexity classifications despite the added uncertainty of limited visibility. Experimentally, our Python-based tool validates these theoretical insights: while manual strategy verification executes almost instantaneously, automated strategy generation exhibits significant runtime increases as complexity bounds grow, a trend clearly demonstrated in our case study on gossip protocols. Future work will focus on further optimizing our approach. In particular, we plan to integrate machine learning techniques into the automatic strategy generation process to improve efficiency and reduce execution times. Moreover, we aim to extend our tool to handle more complex LTL formulas by incorporating automata-based approaches, thereby expanding its verification capabilities. We also plan to extend our approach to support recall-based strategies.

References

1. Aruta, M., Malvone, V., Murano, A.: Development of natural strategies in strategic logics (short paper). In: RCRA Workshop on Experimental Evaluation of Algorithms, co-located with AIxIA 2024. CEUR Workshop Proceedings, vol. 3883 (2024), https://ceur-ws.org/Vol-3883/paper1_SPIRIT_Aruta.pdf
2. de Boer, F.S., Hindriks, K.V., van der Hoek, W., Meyer, J.J.C.: Agent programming with declarative goals. Proceedings of ATAL p. 228–243 (2000), https://api.semanticscholar.org/CorpusID:1834213
3. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Workshop on logic of programs. pp. 52–71. Springer (1981)
4. Gutierrez, J., Harrenstein, P., Wooldridge, M.: Iterated boolean games. Information and Computation **242**, 53–79 (2015). https://doi.org/https://doi.org/10.1016/j.ic.2015.03.011, https://www.sciencedirect.com/science/article/pii/S0890540115000267
5. Gutierrez, J., Harrenstein, P., Wooldridge, M.: Reasoning about equilibria in game-like concurrent systems. Annals of Pure and Applied Logic **168**(2), 373–403 (2017). https://doi.org/https://doi.org/10.1016/j.apal.2016.10.009, https://www.sciencedirect.com/science/article/pii/S0168007216301324, eighth Games for Logic and Programming Languages Workshop (GaLoP)
6. Gutierrez, J., Najib, M., Perelli, G., Wooldridge, M.J.: EVE: A tool for temporal equilibrium analysis. In: Automated Technology for Verification and Analysis - 16th Int. Symposium, ATVA 2018. LNCS, vol. 11138, pp. 551–557. Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_35, https://doi.org/10.1007/978-3-030-01090-4_35
7. Jamroga, W.: Logical Methods for Specification and Verification of Multi-Agent Systems. Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland (2015)
8. Jamroga, W., Malvone, V., Murano, A.: Natural strategic ability. In: Artificial Intelligence. vol. 277, p. 103170. Elsevier (2019). https://doi.org/https://doi.org/10.1016/j.artint.2019.103170
9. Jamroga, W., Malvone, V., Murano, A.: Reasoning about natural strategic ability. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems. p. 714–722. AAMAS'17, IFAAMAS (2017)
10. Jamroga, W., Malvone, V., Murano, A.: Natural strategic ability under imperfect information. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. p. 962–970. AAMAS'19, IFAAMAS (2019)
11. Markey, N., S.P.: Model checking a path. Amadio, R., Lugiez, D. (eds) CONCUR 2003 - Concurrency Theory. CONCUR 2003. LNCS, vol 2761. (2003), https://link.springer.com/chapter/10.1007/978-3-540-45187-7_17
12. Osborne, M., Rubinstein, A.: A Course in Game Theory. MIT Press (1994)
13. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (SFCS 1977). pp. 46–57 (1977). https://doi.org/10.1109/SFCS.1977.32
14. Toumi, A., Gutierrez, J., Wooldridge, M.: A tool for the automated verification of nash equilibria in concurrent games. pp. 583–594 (10 2015). https://doi.org/10.1007/978-3-319-25150-9_34
15. Wooldridge, M., Gutierrez, J., Harrenstein, P., Marchioni, E., Perelli, G., Toumi, A.: Rational verification: From model checking to equilibrium checking. In: AAAI Conference on Artificial Intelligence (2016), https://api.semanticscholar.org/CorpusID:17512723