Model Checking Nash Equilibria in Probabilistic Games

ABSTRACT

Verifying whether rational participants in a probabilistic distributed system would deviate from the specified behavior is an important but challenging task. We term this problem as verifying nash-equilibria in probabilistic cooperative distributed multi-player games. Existing algorithms only work for non-probabilistic systems. We develop a model checking algorithm for automatically verify nash-equilibria for probabilistic systems. The algorithm is implemented based on a model checker - PAT (Process Analysis Toolkit). Using the algorithm and our tool, a case study has been performed on analyzing the first probabilistic secret sharing scheme.

CCS Concepts

•**Software and its engineering** → **Model checking;** •**Mathematics of computing** → *Probability and statistics;* •**Theory of computation** → *Algorithmic game theory and mechanism design;* •**Security and privacy** → Security protocols;

Keywords

Model Checking; Nash Equilibria; Probabilistic Game; Contract Signing; Secret Sharing

1. INTRODUCTION

Many systems in the real-world involve collaboration of many distributed parties, e.g., Internet routing [26], peer-to-peer file sharing [3], cooperative backup [18], etc. In many of such systems, participants may have conflict interests. Thus, it is natural to consider each party as rational, meaning that each party only collaborates if his benefit from collaborating is bigger than not collaborating. That is, the system is actually a game. An important property for such systems/games is to ensure that every rational player would not deviate from the expected behavior specified in the systems. That is to say, to pursue the biggest benefit, the rational players would all follow the system, i.e., the system is the nash-equilibrium in the concept of game theory – no one derivate from the system because this would harm their benefits.

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

Traditionally, when designing a system, participants are assumed to be either "good guys" – following the system behavior, or "bad guys", who do everything in their power to break the system [1]. Rational/selfish participants are seldom considered; and few systems[1] work correctly under the rational settings (as stated in [22]).

When the desired properties, such as security or reliability, of such systems is critical, whether every collaborator follows the system becomes of great importance. In this case, precisely proving that the system is indeed the desired equilibrium is necessary. In the literature, such proofs are often manual, e.g., proofs in the contracting signing protocol [24] and the secret sharing protocol [13]. As the proof tasks are often heavy, manual proof is error-prone and inefficient, especially the designer of a system may not be an expert on formal proofs. Thus, automated proof is desirable. Model checking, which is a precise way to verify systems and has many automated tool support, is a promising method.

Automated model checking tools for games exist, e.g., MOCHA [2], MCMAS [19], GAVS+ [7] and PRISM-games [6]. The existing tools verify various types of properties in various types of games. However, none of them is able to verify the nash-equilibrium type of property – whether following the system is the optimal strategy for each rational players, in concurrent games. Recently, an algorithm is developed to verify such properties in a sub-class games which can be described using finite state mechanisms [22]. However, this work only considered games without probability. In real-world, many systems have probabilistic behavior: Some of the systems are inherently probabilistic, for example, economic and biology; while some other systems fail to achieve certain goal without involving probability, for example, it is proved that non-probabilistic secret sharing scheme is not practical in the rational settings [13]. Hence, we extend their algorithm to verify nash-equilibrium for probabilistic systems.

Contributions..

We develop a verification algorithm to verify whether a probabilistic system is a nash-equilibrium. We implement the algorithm based on PAT (Process Analysis Toolkit), which is a model checking platform providing efficient general probabilistic model checking algorithms [23]. We perform a case study on analyzing a probabilistic secret sharing scheme [13]. The authors of the scheme first prove that no practical mechanism for secret sharing works without using probability in the rational setting, then provide the first secret sharing with the rational participants in mind. This case study first illustrates how our algorithm and tool help ease the proofs of the non-existence of non-probabilistic scheme. Second, it shows how the tool automatically verifies the correctness of the probabilistic scheme. In addition, during verification, weaknesses of the proba-

[1]Notable exceptions include [1, 17]

bilistic scheme have been found.

2. RELATED WORKS

Nash-equilibria is a popular topic in game theory. Algorithms exist for finding nash-equilibria in games [11, 9, 8]. However, these works usually use simulation and mathematical analysis techniques [4]. Model checking has the advantages of both simulation, being automatic, and mathematical analysis, coving all possible behaviors. Hence, model checking has been applied to verify game theoretic properties, for example [14, 10, 20]. Among these works, we highlight the following which focused on probabilistic behaviors and automatic model checking tools for games.

The model checker MOCHA [2] has been widely used for verifying games, e.g., [15, 21, 28]. However, nash-equilibria checking is not explicitly supported, and probabilistic behaviors is not supported. Recently, a tool PRALINE is developed specifically for computing nash-equilibria [5], and a tool EAGLE is developed for verifying nash-equilibria for concurrent games [27]. However, these tools do not support probability.

Moder checker PRISM [16] supports probabilistic behaviors, but does not explicitly support verifying game properties. Using the Rubinstein's protocol as a case study, Ballarini et al. show that probabilistic model checking (using the model checker PRISM) can be used for game analysis [4]. This work inspirits automated probabilistic model checking for games, but it is specific to the negotiation game (a bargaining between a buyer and a seller) framework and does not address nash-equilibria analysis. Later, PRISM-Games extends the model checker PRISM to be able to verify turn-based games [6]. PRISM-game is able to verify nash-equilibriums, but only for turn-based games.

The most relevant work is the one by Mari et al. [22]. This work proposes a symbolic model checking algorithm for verifying nash-equilibrium in distributed cooperative systems. It it the first algorithm to automatically verify that it is in the best interest of each rational agent to follow exactly a given system [22]. A player's all possible behavior is modeled as a finite state mechanism. The given system describes the which action should be taken in which situation (by honest/altruistic players), which is modeled as constrains on state transitions of a game. Infinite sequences of actions is tackled by using *discount factor* [12] to decrease relevance of rewards that are far in the future, which allows them to only look at finite sequences of actions. The authors propose a notion of equilibrium which takes both *Byzantine* players – players that randomly take possible actions and ignore utilities/rewards, and a *tolerance* of rewards – the reward difference between following the system and deviating from the system, as parameters; and prove that using their algorithm the proposed equilibrium can be automatically verified by just looking at finite sequences of actions. This algorithm does not support verification of probabilistic systems. We adopt a similar specification framework of players and equilibrium, and develop nash-equilibria verification algorithm, which focuses on probabilistic systems.

3. SPECIFICATION

As mentioned in the introduction, we consider distributed systems with multiple players who need to cooperate in order to achieve a goal. Games for such systems are inherently distributed with multi-players, rather than turn based. In addition, a system can be non-terminating, i.e., there are infinite executing steps for some players. For instance, a system (e.g., the case study protocol) can be executed unbounded number of times.

Games are often represented in a normal form – a matrix of players, strategies and payoffs, or in the extensive form – a finite game tree specifying the sequences of actions. Such specification is no longer suitable when the game has infinite execution steps. To be able to model infinite games, we adopt the fine state mechanism representation defined in [22] – each player is specified as a finite state mechanism, potentially with loops. This allows us to specify an infinite execution steps in a finite manner.

We first briefly introduce the adopted specifications in Section 3.1, and then explain in details the probabilistic extension in Section 3.2. Finally, we use an example to illustrate how a probabilistic system is specified.

3.1 Preliminaries

We use the formalism of finite state mechanisms [22] to model all possible local states and local actions of players. All players' local states and actions together form all possible global states and actions in a game. The probability to perform the actions will be added in Section 3.2.

Formal Definition of Finite State Machine.

Each player i has a set of local states, denoted by S_i, a set of initial states, denoted by I_i, and a set of actions, denoted A_i. The global states in a game with n players is the product of the local states of each player, i.e., $S = S_1 \times S_2 \times \cdots \times S_n$; the set of initial states is $I = I_1 \times I_2 \times \cdots \times I_n$; and the set of global actions is $A = A_1 \times A_2 \times \cdots \times A_n$.

Function $B_i : S \times A_i \times S_i \to \mathbb{B}$ is used to specify the feasibility of player i performing a action $a \in A_i$, which leads i's local state to $s' \in S_i$, at a global state $s \in S$ – if it is feasible, $B_i(s, a, s')$ is true, otherwise false. The function B_i is required to have two properties: *serial* and *deterministic*:

- Serial property says that there is at least one action defined for each player at each global state, formally, $\forall s \in S, \exists a \in A_i, \exists s' \in S_i$ s.t. $B_i(s, a, s')$ is true;
- the deterministic property says that a player's action a should lead to a unique resulting local state, formally $(B_i(s, a, s_1) = true \wedge B_i(s, a, s_2) = true) \implies s_1 = s_2$.

Due to the deterministic property, we simply write $B_i(s, a)$ to represent that action a is valid in a global state s, since the local state s' is uniquely defined. Similarly, the global possible actions of all players are $B = B_1 \times B_2 \times \cdots \times B_n$.

In non-probabilistic settings, $\langle S_i, I_i, A_i, B_i \rangle$ defines a Byzantine player, since a Byzantine player can perform all actions that are feasible at a state, ignoring utility/rewards. Recall that a system specifies which action should be performed at which state by which player, i.e., a system is a specification of altruistic players. Similar to B_i, function $T_i : S \times A_i \to \mathbb{B}$ is defined to specify a player i's behavior of genuinely following the system – $T_i(s, a) = true$ ($s \in S$, $a \in A_i$) if i shall perform a at state s according to the system. Since following the system (altruistic player) is always part of the possible behavior, we have $T_i(s, a) \to B_i(s, a)$. In addition, we assume the system does not have unspecified situation or deadlocks. Formally, we require that T_i is non-blocking (i.e. $\forall s \in S, \exists a \in A_i$, s.t. $T_i(s, a)$). A system specifies the altruistic players, formally specified as $T = \langle T_1, \cdots, T_n \rangle$.

Since that function T_i models the altruistic player's behavior, and B_i models a Byzantine player's behavior, given a set of Byzantine players (denoted as Z) among the total n players, the behavior of Byzantine and altruistic players is summarised as:

$$BT(Z, s, a_i, s') = \begin{cases} B_i(s, a_i, s') & i \in Z \\ B_i(s, a_i, s') \wedge T_i(s, a_i) & i \notin Z \end{cases}$$

In probabilistic settings, in addition to the feasible actions, a probability is given to each action.

Payoff Function.

Differing from Byzantine players and altruistic players, a rational player's behavior is driven by rewards, a.k.a. payoffs. Player i's payoff is defined as a function $h_i : S \times A \to \mathbb{R}$, i.e., for each global state transition and a global action, there is a payoff value for player i, denoted as numbers in $\mathbb{R}$. Hence, $h = \langle h_1, \cdots, h_n \rangle$ denotes the payoff of all the n players at any state for performing any action. The payoff function can be directly adopted to the probabilistic settings since it is not influenced by adding probability.

Discount factor.

Let $\beta = (\beta_1, \cdots, \beta_n)$ be the n-tuple of discount factors for the n players, where $\beta_i \in (0, 1)$ is the discount factor for player i. The discount factor is used to decrease the relevance of rewards decreases in the future, so that to ensure efficient computation of a finite strategy in the settings of infinite games. We adopt the discount factor as well in order to handle infinite games in probabilistic settings.

Path.

With the above definitions, a non-probabilistic game can be specified as $(S, I, A, B, T, h, \beta)$. Given such a game with a set of Byzantine Z, a path is defined as a sequence of possible (s, a) pairs where $s \in S$ and $a \in A$. We denote a path as $X = \langle X_0, \cdots, X_n, \cdots \rangle$, where each step X_t ($t \geq 0$) is a (s, a) pair. We use $X_t^{(s)}$ to denote the first element of a step in a path – the state s and $X_t^{(a)}$ to denote the second element – the action a. A path is valid if $\forall t$, $BT(Z, X_t^{(s)}, X_t^{(a)}, X_{t+1}^{(s)}, X_{t+1}^{(a)})$ is true, meaning that the the transition from one step to the next step is valid.

The utility of a player i in a path is evaluated by adding up the discounted payoff values along the path, formally, $V_i(X) = \Sigma_{t=0}^{|X|-1} \beta_i^t h_i(X_t^{(s)}, X_t^{(a)})$, where $|X|$ is the path length, which can be infinite. When the path length is infinite, $V_i(X)$ converges to an upper bound, because of β_i.

Strategy.

To reason on a game, it is often needed to distinguish one player's actions from other players. Hence, the notion of strategy is defined as follows: A strategy of a player is a sequence of local actions of the player. A strategy at a state s defines what a player would behave in the future. Depending on other players' actions, there will be multiple paths with multiple utility values. The minimum utility value is called the *guaranteed outcome* of the strategy for the player at the state s. In the algorithm in [22], two important values are calculated:

- the *value* of a state s at horizon k for player i with Byzantine players Z, denoted as $v_i^k(Z, s)$, is defined as the guaranteed outcome of the best strategy of length k starting ate state s;
- the *worst case value* of a state s ate horizon k for player i with Byzantine players Z, denoted as $u_i^k(Z, s)$, is defined as the guaranteed outcome of the worst strategy of length k starting at state s.

We show, in the next section, how these two values are calculated in the probabilistic settings.

3.2 Probability Extension

We extend the above specification to be able to specify probabilistic systems. A probabilistic system specifies which actions of a player is probabilistic and specifies a probability for each actions. To be more general, we allow hybrid probabilistic systems, meaning that some actions of a player in the systems are probabilistic, some of the actions can be deterministic and some others can be non-deterministic.

First of all, we update the model of players. Then we extend the state representation to be probabilistic. More importantly, we present the calculation of the value $v_i^k(Z, s)$ and worst value $u_i^k(Z, s)$ of a state in the probabilistic settings.

Byzantine Player.

In a non-probabilistic setting, a Byzantine player is modeled as all possible local states and their corresponding feasible local actions. In a probabilistic setting, a Byzantine player has exactly the same local states and corresponding local actions, since probabilistic systems do not introduce additional local actions. That is to say, similar as in the non-probabilistic setting, a Byzantine player can deviate from a system specification by performing actions that are not defined in the system. In addition to perform extra actions, for the probabilistic actions in a system, a Byzantine player is able to deviate from the specification by changing the probability distributions of the actions. For example, if a system specifies a player's action as follows: the player toss a coin - with probability $1/2$ to be head and $1/2$ to be tail, if it is head, the player sends 1, otherwise, sends 0. A Byzantine player can deviate from the specification by changing the coin into a special one – with probability $1/3$ to be head and probability $2/3$ to be tail. Furthermore, a Byzantine player is able to assign probability to non-probabilistic actions. For example, a system specifies a deterministic action of sending 1, a non-probabilistic Byzantine player can deviate by sending 0, a probabilistic Byzantine player can toss a coin – with probability α to be head and probability $1 - \alpha$ to be tail, and sends 1 if it is head and sends 0 if it is tail. To summarize, a Byzantine player is able to assign an arbitrary probability distribution (which adds up to 1) to at a local state.

Since local actions at a local states are feasible actions at the corresponding global states, this can be formally modelled by defining a function $P_i : S \times A_i \to \mathbb{R}$ where $\mathbb{R}$ ranges over $[0, 1]$, with the requirement that at a global state s, $\Sigma_{a \in A_i \wedge B_i(s,a)} P_i(s, a) = 1$.

Altruistic Player.

An altruistic player follows system specification, and thus has predefined probability for each probabilistic action and may have deterministic and non-deterministic actions depending on the system specification. Locally, this can be modelled by adding the predefined probability to probabilistic actions and keeping other actions the same as in the non-probabilistic settings.

Globally, in order to specify the hybrid actions, we need to distinguish which actions are probabilistic and which are not. Since the system specifies the distinction. At a global state, we are able to tell the probabilistic actions from the non-probabilistic actions. We use $PA_i(s)$ to denote the probabilistic actions of player i at global state s. We require that $PA_i(s) = \{a | a \in A_i \wedge T_i(s, a)\}$ and $\Sigma_{a \in A_i(s)} P_i(s, a) = 1$. This definition only applies to the altruistic behaviour of a player. The former requirement means that the probabilistic actions should be one of the possible actions and is valid at state s. The later requirement says that the probability of all probabilities actions adds up to 1. Define $PA = (PA_1, PA_2, \cdots, PA_n)$.

df: Will assume that all the actions are either probabilistic or deterministic for honest players

Rational Player.