Model Checking Nash Equilibria in Probabilistic Games

Dileepa Fernando
Supervisor - A/P Jin Song Dong
School of Computing,
National University of Singapore.

ABSTRACT

Verifying whether rational participants in a probabilistic distributed system would deviate from the specified behaviour 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, we performed a case study on analysing the first probabilistic secret sharing scheme. Algorithm was evaluated by applying to a continuous task sharing scenario.

1. INTRODUCTION

Many systems in the real-world involve collaboration of many distributed parties, e.g., Internet routing [40], peer-to-peer file sharing [5], cooperative backup [30], 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 more than not collaborating. In this sense, 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 most 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.

Traditionally, when designing a system, participants are assumed to be either "good guys" – who follow the system behavior, or "bad guys", who do everything in their power to break the system [2]. Rational/selfish participants are seldom considered, and few systems[1] work correctly under the rational settings (as stated in [35]).

When the desired properties, such as security or reliability, of such systems are critical, whether every collaborator following the system becomes of great importance. In this case, precisely proving that the system is indeed the desired equilibrium is necessary. Such proofs are often manual, e.g., proofs in the contracting signing protocol [38] and the secret sharing protocol [22]. 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[6], 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 [4], MCMAS [31], GAVS+ [14] and PRISM-games [12]. 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 has been developed to verify such properties in a sub-class games which can be described using finite state mechanisms [35]. However, this work only considered games without probability. In reality, many systems have probabilistic behaviours. Some of the systems are inherently probabilistic. Economic and biology systems are examples. There are other systems that fail to achieve certain goals without involving probability, for example, it is proved that non-probabilistic secret sharing scheme is not practical in the rational settings [22]. Hence, we propose an approach to verify nash-equilibrium for probabilistic systems.

Contributions..

We develop a verification algorithm to verify whether a probabilistic game 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 [37]. We perform a case study on analyzing a probabilistic secret sharing scheme [22]. The authors of the scheme first prove that no practical mechanism for secret sharing works without using probability in the rational setting and further provide the first secret sharing with the rational participants in mind. This case study first illustrates how our algorithm and tool help to 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 probabilistic scheme have been found.

2. RELATED WORK

Nash-equilibria is a popular topic in game theory. Algorithms exist for finding nash-equilibria in games [19, 17, 15]. However, these works usually use simulation and mathematical analysis techniques [7]. Model checking has the advantages of both simulation, being automatic, and mathematical analysis, covering all possible behaviors. Hence, Model Checking has been applied to verify game theoretic properties, for example [24, 18, 32]. Among these works,

[1]Notable exceptions include [2, 29]

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

we highlight the following which focused on probabilistic behaviors and automatic model checking tools for games.

The model checker MOCHA [4] has been widely used for verifying games, e.g., [26, 33, 42]. However, nash-equilibria checking is not explicitly supported, and probabilistic behaviours are not supported. Recently, a tool PRALINE has been developed specifically for computing nash-equilibria [10], and a tool EAGLE is developed for verifying nash-equilibria for concurrent games [41]. However, these tools do not support probability.

Model checker PRISM [28] supports probabilistic behaviours, 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 [7]. 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 [12]. PRISM-game is able to verify nash-equilibriums, but only for turn-based games.

Probabilistic model checking has been used to analyse multi agent dynamics [23] . Two strategies to arrive at maximal dispersion outcome (MDO) [23] in a generalized anti-coordination game has been discussed. Counter Abstraction is used to reduce the state space. MDO can be seen as some form of equilibrium of the multi agent game, but it is not the formal Nash Equilibrium. Hence, this work is an application of probabilistic model checking to a specific class of games and it does not generally address the concept of nash-equilibrium.

The most relevant work is the one by Mari et al. [35]. This work proposes a symbolic model checking algorithm for verifying nash-equilibrium in distributed cooperative systems. It is the first algorithm to automatically verify that it is in the best interest of each rational agent to follow exactly a given system [35]. A player's all possible behaviour is modelled as a finite state mechanism. The given system describes which action should be taken in which situation (by honest/altruistic players), which is modelled as constrains on state transitions of a game tree. Infinite sequences of actions are tackled by using *discount factor* [20] 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. Even though we define the verification algorithms for distributed systems with cooperative actions, the algorithm can be easily extended to support distributed systems with turn based actions as well.

3. BACKGROUND

3.1 Game Theory

Game theory is a wide area of research which is used in various areas such as economics, computer architecture, computer security and robotics. Two main aspects of a scenario modelled in game theory are the well defined utility and the information, each player has [36]. For a game theory problem, there has to be more than one party involved.

In a broad sense, games theory categorizes four kinds of games namely strategic games, extensive games with perfect information, extensive games without perfect information and coaliational games.

Strategic Games.

[36] In a strategic game we have a set P of players and each player P_i has a non empty set of actions A_i. Each P_i has a preference relation $\geq_i$ defined in $A_1 \times A_2 \times \cdots A_n$ which denotes how a player is affected from different action combinations. An action is denoted $(a_1, a_2, \cdots, a_n)$. As a short notation to compact the action combination of the players other than P_i we use a_{-i}. The action combination $(a_1, a_2, \cdots, a_n)$ can also written as (a_{-i}, a_i) when we want to explicitly differentiate between player P_i's actions and actions of the other players. This notation can be extended to denote the action combinations of player sets. For a set st, $(a_{st}, a_{[n]-st})$ denotes the corresponding actions are categorized w.r.t the player set.
A Pay off function is defined for a special case where the preference relation is a total order.

Extensive games with perfect information.

[36] These kind of games have a set P of players and a set of sequences (finite or infinite) H which stand for the history of actions of a game. A member $h \in H$ is called terminal (history towards end of a game) if h is infinite or not a prefix of any other member in H. We use Z to denote the set of all terminal sequences in H. A preference relation (similar for strategic game) is defined for these terminal sequences. For all the non terminal sequences in H a function $Turn : H \setminus Z \rightarrow P$ specifies the player who has the next move. A player can choose between a set of actions depending on his history h. Here perfect information refers to the fact that at any point of the game all the players have history information.

Extensive games without perfect information.

[36] In contrast to extensive games with perfect information, this type of games have a probability distribution for the set of actions to perform given a history h. In addition to the probability, there are information partitions defined for each player. All the history sequences where the resulting action set is similar are categorized into same partition. When a player does not know his exact history but knows all the possibilities for history and his action set can short list the possible history values according to information partitions.

Coalitional games.

Coalitional games are described for the completeness. Coalition games have a set P of players and it has a value function $V : \mathbb{P}P \setminus \phi \rightarrow \mathbb{R}$ to define the pay off value obtained for each coalition.

Nash Equilibrium.

[36] Nash Equilibrium is a state of a game ending in which no player has a regret on the strategy he used when he cannot control the strategies of other players. We define a Nash equilibrium for a strategic game as follows.

DEFINITION 1. *For a strategic game of player set P and action set $A_1 \times A_2 \times \cdots \times A_n$ and preference relation $\geq$ a nash equilibrium is defined as an action combination $a^* \in$ such that,*
$\forall P_i \in P$
$(a^*_{-i}, a^*_i) \geq_i (a^*_{-i}, a_i) \forall a_i \in A_i$

It is not always the case that, a nash equilibrium according to above

definition exists. However, for any strategic game, a mixed strategy Nash equilibrium exists. Nash Equilibrium for other forms of games also have the same concept of best response. We will not discuss all the definitions here, and we will refer to specific Nash Equilibrium definitions in section 7.

In computer science research, the most important aspect we look in game theory is, computation and verification of Nash equilibria. The computation complexity is highly dependant with the number of strategies. Calculating Nash equilibrium in strategic games is PPAD complete. [16],[13].

Extensive form games are widely used to model actual games in real life where the players take turns. There are some applications that can be regarded as games and can be represented by non of the above game categories. For example, processes/protocols can be defined as finite state mechanisms. However, the closest representation of the above mentioned representations is extensive form. This representation itself is not very efficient.

3.2 Model Checking

It is essential to validate a software system against its requirement specification. For complex systems, these properties has to be verified before the system development. Requirements can be mathematically defined as properties while actual systems can be presented as mathematical models [6]. Different system behaviours are referred to as models and the property verification procedure is called “Model checking".

The properties to verify include deadlock freeness, starvation freeness, etc. Some systems may allow some degree of misfunctionality. By using model checking algorithms we can find/verify the error bounds, maximum error probability, etc. Properties can be categorized into safety properties and liveliness properties. NASA's deep space-1 space craft and validation of the execution engine of the core TM i7 processor[25] are some of the successful usages of model checking.

The first step of model checking procedure is to specify the system model and the property to be verified in the language of model checking. Then simulations can be performed to have some confidence about the operation of the system model. Next step is to check whether the property is satisfied by the system. If the property does not hold then model checking tool can generate a counter example. Based on the counter example, the system model can be refined iteratively until the property holds.

A major challenge of model checking is state space explosion in real time system models. Researchers in model checking have been working on reducing the state space by using various abstraction techniques such as binary decision diagrams[35] and partial order reduction[3]. These abstractions can be application specific or generic. Abstractions may also be property specific.

Verifying game theoretic properties in protocols has become very important and tricky because, the original definitions of game representations and the generic algorithms to find the Nash equilibriums do not help a lot in optimizing protocol verification. As mentioned in the previous section, model checking research community strongly focuses on application and property specific optimizations. Developing model checking algorithms to verify game theoretic properties is very useful under this background.

On the other hand, some real world protocols are easy to analyse when they are represented as games. Following section provides some examples of using model checking to verify game theoretic properties.

4. APPLICATIONS OF MODEL CHECKING FOR GAME THEORY

Contract signing protocols have game-like properties that can be verified by model checking. A two-party contract is an exchange agreement.Informally, if both parties perform the contract, they have some utility. If one party violates the contract, the violator gets more utility and the other party receives less. It is evident that designing a simultaneous contract signing protocol which is fair, without either a trusted third party or a partial commitment scheme is usually difficult in practice.[8] A simple example for a partial commitment scheme is as follows.

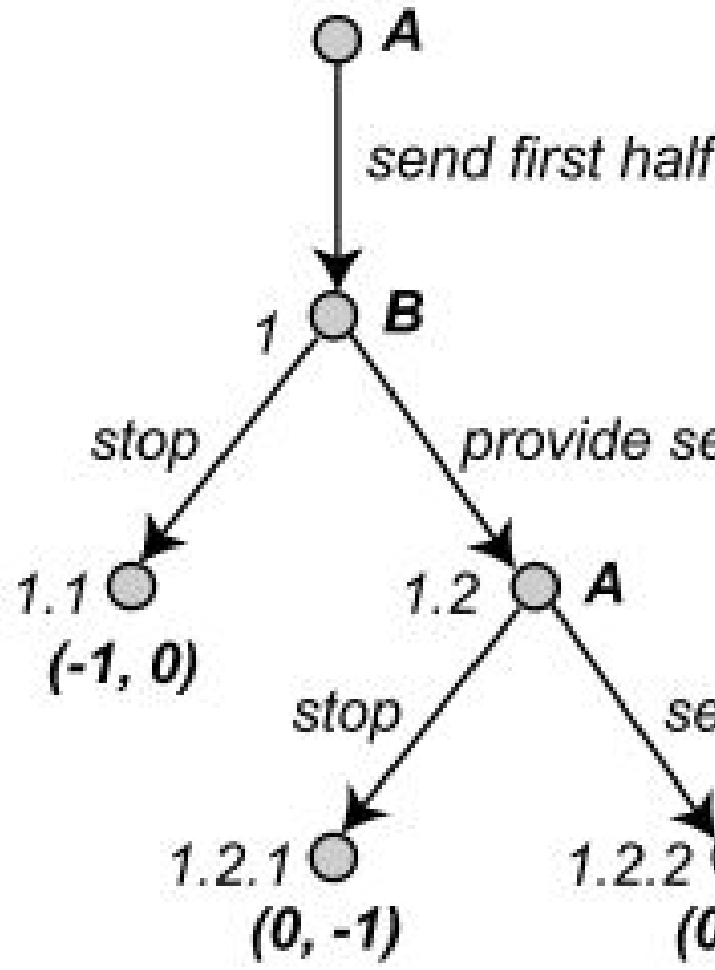


Figure 1: Exchange Example

Figure 1 [11] defines a protocol in which player A sends a half coin before player B provides the service. If player B provides the service, player A can send the other half of the coin. If player B does not provides the service, player A gets a negative reward, because he lost half a coin and he has another half of a coin which he can't use. If player A does not send his first half of the coin, protocol does not proceed and both players have no loss nor gain. The utilities for A and B are shown within brackets. This is exactly an

example for an extensive game with perfect information. A similar example about a certified email protocol is mentioned in [11].
Probabilistic protocols for contract signing introduces parameters quantify fairness. Rabin's protocol[38] and BGMR protocol[8] are two such probabilistic contract signing protocols. Following are the major steps of Rabin's protocol.

Protocol Rabin's Protocol

1: $\forall k$ Let $R(k)$ be a random number $\in \{1, 2, \cdots, N\}$
2: let A , B going to decide on contract C
3: Decide on dead line D
4: **for** $i = 1$ to N **do**
5: $S =$ "I am committed to C if $i = R(D)$"
6: $M = sig_A(S)$
7: $A \rightarrow_M B$
8: $S =$ "I am committed to C if $i = R(D)$"
9: $M = sig_B(S)$
10: $B \rightarrow_M A$
11: **end for**
12: Generate $R(D)$

if the players follow the protocol they both commit to the contract with a probability 1. If we assume that authenticity of the messages are preserved, avoiding sending of the messages is the only way to cheat. There is a strategy with probability $\frac{1}{N}$ to make an honest player committed while a rational player is not committed [38].
BGMR[8] protocol does a slight modification to statements 5 and 8 by replacing the commitment probability which is increased over the iterations. Model checking can be used to verify and find cheating strategies in the above protocols [1]. Usage of model checking occurs in these examples in the verification part.
Formal definitions were introduced to specify fair-exchange/contract signing protocols as games. These definitions can be flexible to specify different fairness definitions and would be useful for formal verification [11]. They define the concepts of fairness, safe fairness and Nash equilibrium fairness and how a fair-exchange game operates under imperfect information.

Modelling languages are defined to specify protocols as games [27] discusses about modeling non repudiation protocols as games (ATL) and verifies these protocols by using MOCHA.Proposed future work were verifying abuse free-ness in contract signing protocols and investigating strategy synthesis.

5. MOTIVATION

One research problem in game like protocols/systems is the need to find a most efficient representation for the analysis of game theoretic properties. Designing new modelling language constructs to automatically generate the game representation by the system model is highly essential. Another research problem is finding verification and synthesis algorithms based on the representation. If proposed algorithms are time consuming, approximate algorithms can be introduced.
We consider verifying the rationality of finite state concurrent systems. We restrict our analysis to extensive games with prefect information. To be more general, we would assume the system/protocol choices be probabilistic.

6. 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 an extensive form – a finite game tree specifying the sequences of actions. Such specification is no longer suitable when the game has infinite execution steps. In order to model infinite games, we adopt the finite state mechanism representation defined in [35] – each player is specified as a finite state mechanism, potentially with loops. This allows us to specify infinite execution in a finite manner.

We use the formalism of finite state mechanisms [35] 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.

Our contribution in this section is, extension of the finite state mechanism in [35] for probabilistic settings. We describe, how we changed the non-probabilistic representation to arrive at a probabilistic representation. First of all, we update the model of players to be probabilistic. 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.

We start from general properties of players and end with specific properties of player types. 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. Action set representation would follow from the definitions in section 3.1.

Set of all the local actions can be denoted by $\hat{A} = \cup A_i$. Before defining the transition function we must define different player behaviours. We also introduce synchronized actions. The set of synchronised actions can be denoted by A^s. $A^s \subseteq \hat{A}$

Byzantine Player.

In a non-probabilistic setting, a Byzantine player is modelled 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 the 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.

Altruistic Player.

An altruistic player follows system specification, and thus has

predefined probability distribution for each probabilistic action. A probabilistic altruistic player covers the definition of non-probabilistic altruistic player as well. Each state has a unique probabilistic action defined. If only one action is defined it can be considered as a specific case of containing a single action of probability 1.

Since local actions at a local states are feasible actions at the corresponding global states,

We model probabilities of performing local actions starting at given local states. Define $P_i : S_i \times A_i \to [0, 1]$, with the requirement that $\forall s_i \in S_i, \Sigma_{a_i \in A_i} P_i(s_i, a_i) = 1$.

A local action a_i is defined in local state $s_i \iff P_i(s_i, a_i) > 0$.

Rational Player.

A rational player is a special case of a Byzantine player who has the full power to change the probability distribution of actions at a state, driven by utility. We show that a rational player would always choose an action which gives the maximum utility with probability 1. Taking the simplest case with two actions,

- If both of the actions lead to paths that give exactly the same utility, the probability distribution of the two actions does not influence the utility. Hence, we can assume the player choose one action with probability 1, the other with probability 0, without loss of generality.
- If the two actions lead to paths that give different utility, the player would choose the one gives bigger utility with probability 1, the other with probability 0.

The same reasoning holds for the case that the player has more than two feasible actions. Therefore, we can safely model a rational player with non-probabilistic actions.

Local State Transition.

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, i.e. ($\forall s \in S$, $\exists a \in A_i, \exists s' \in S_i$ 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, i.e. ($(B_i(s, a, s_1) = true \wedge B_i(s, a, s_2) = true) \implies s_1 = s_2$.)

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 as well as 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 behaviour 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 behaviour, we have $T_i(s, a) \implies B_i(s, a)$. 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 behaviour, and B_i models a Byzantine player's behaviour, given a set of Byzantine players (denoted as Z) among the total n players, the behaviour of Byzantine and altruistic players is summarised as:

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

Payoff Function.

[35]

Player i's payoff is defined as a function $h_i^{\psi} : S_i \times A_i \to \mathbb{R}$, i.e., for each global state transition and a global action, there is a payoff value for player i, denoted as numbers in $\mathbb{R}$. Here, ψ is a set of global constraints. 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 $\beta_i \in (0, 1)$ 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.

Local Finite State Machine.

The local finite state machine for player i, in a probabilistic setting can be defined as $(S_i, I_i, A_i, BT_i, h_i, P_i)$

We can see how the local state machines can be combined to come up with the global state machine. $(S, I, A, h) = (\Pi_i S_i, \Pi_i I_i, \Pi_i A_i, \Pi_i h_i)$.

The remaining components are P_i and BT_i.

We can define probabilities of global actions given a global state. Global state $s \in S$ can be written as a local state sequence, $\langle s_1, s_2, \ldots s_n \rangle$ and global action $a \in A$ can be written as local action sequence $\langle a_1, a_2, \ldots a_n \rangle$ as defined above. Now we can define, Probability function for global actions $P : S \times A \times \mathbb{P}([n]) \to [0, 1]$.

$$P(s, a, Z) = \prod_{i \notin Z} P_i(s_i, a_i)$$

The global transition function BT also should be updated to support synchronized actions. We define set J_i ,for $i \in 1 \ldots n$ to count the number of occurrences of synchronized local action a_i in the action sequence a.

$$J_i(a) = \{k \,|\, k \in 1 \ldots n \wedge a_i \in A^s \wedge a_i = a_k\}.$$

$$BT(Z, s, a, s') = \begin{cases} \wedge_i BT_i(Z, s, a_i, s_i') & \forall\, i, a_i \notin A^s \\ \wedge_i BT_i(Z, s, a_i, s_i') \wedge (J_i(a)! = 1) & otherwise \end{cases}$$

It should be noted that if $BT(Z, s, a, s') = false$ then s' is a deadlock state. Now, summarized global finite state graph in the probabilistic setting is defined by, (S, I, A, BT, h, P).

We can move on to value calculations on the global finite state graph. We would need two concepts called path and strategy.

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. Path length is the number of global actions contained by the path. A path X is valid if $\forall t < |X|$, $BT(Z, X_t^{(s)}, X_t^{(a)}, X_{t+1}^{(s)})$ is true, meaning that the the transition from one step to the next step is valid.

In non-probabilistic setting, 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.

$Path_k(s, Z) = \{\pi | \pi$ is a path in $(M, Z) \wedge |\pi| = k$ and $\pi^{(s)}(0) = s\}$ In a probabilistic setting, a path denotes a set of paths that are reachable through unique probabilistic strategies of Altruistic players and a strategy of Byzantine players. A path tree can be denoted by $\{\pi\}$. A path tree of length k is denoted by $\{\pi|_k\}$ The concept of a probabilistic path can be defined through the concept of probabilistic action. Each non deterministic action choice by the set of Byzantine players correspond to one path tree in the global graph.

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 [35], 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.

In a probabilistic setting, a strategy is defined the same as in the previous section. The only difference is that all the strategies of an Altruistic player are assigned a probability. Strategies also appear as sets. $\forall pathset\{\pi\}$ there is a corresponding strategy set $\{\sigma\}$ for player i.

In addition to the state machine representation, we would add two definitions for global probabilistic action and a global probabilistic state.

States.

We extend the global state representation in [35] with probability. We can consider resulting states of a probabilistic action as a single probabilistic state.

Actions.

Since some players are probabilistic, a global action is defined as a set of atomic actions where each action can be assigned a probability.

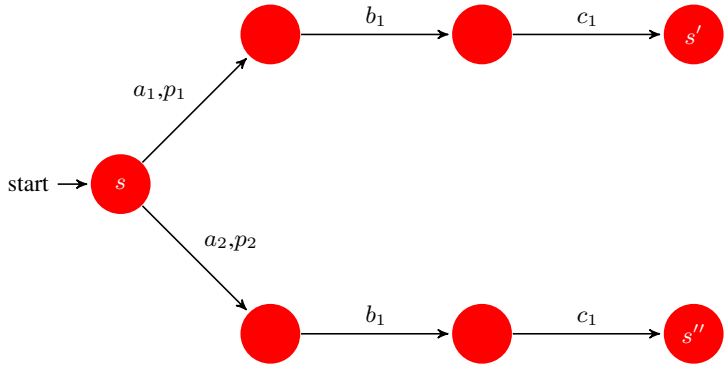


Figure 2: Probabilistic Global Action Example

Let $a \in A$, a_Z be the local action components of Byzantine players and $a_{[n]-Z}$ be the local action components of altruistic players. A probabilistic global action can be defined as,

$GA = \{a \in A | \forall a', a'' \in GA, a'_Z = a''_Z \wedge \cup_{b \in GA} b_{[n]-Z} = A_{[n]-Z} \wedge \forall c', c'' \in GA, c'_{[n]-Z} = c''_{[n]-z} \implies c' = c''\}$

An example action is presented in figure 2. Let $s = \langle s_1, s_2, s_3 \rangle \in S$. Let there are 3 players $1, 2, 3$ with enabled action sets $\{a_1, a_2\}$, $\{b_1, b_2\}$ and $P\{c_1, c_2\}$ respectively. Let 1 be an altruistic player. $P_1(s_1, a_1) = p_1; P_1(s_1, a_2) = p_2$;

This probabilistic action can be written in set notation, $\{(a_1, b_1, c_1), (a_2, b_1, c_1)\}$ The other possible probabilistic actions are, $\{(a_1, b_1, c_2), (a_2, b_1, c_2)\}, \{(a_1, b_2, c_1), (a_2, b_2, c_1)\}$ and $\{(a_1, b_2, c_2), (a_2, b_2, c_2$ same strategy of rational players is applied to both probabilistic choices because of the parallel synchronous nature of the system. Rational players do not have information about the probabilistic choice made by the altruistic player. We extend the following example in [35] with probability, to show how a probabilistic system is modeled.

EXAMPLE 1. *Suppose there are m jobs $\{J_1, \cdots, J_m\}$, and q tasks $\{T_0, \cdots, T_n\}$. Each job is consisted of a set of tasks. A system, consisting of n workers, specifies a sequence of tasks for each worker. A task sequence for player i is denoted as $\mathcal{T}_i = \langle \tau(i, 0), \cdots, \tau(i, l_i) \rangle$, where $l_i + 1$ is the length of the sequence. When finishing one task, a work waits for the rewards. Once the rewards are received, the worker starts the next task. After finishing all tasks in the sequence (i.e., after finishing $\tau(i, l)$), a worker repeats the sequence of tasks from beginning (i.e., task $\tau(i, 0)$). Once a job is completed, the rewards are granted to the works who finished the tasks that compose the job. Assume that tasks for different jobs do not overlap. In completing a job, if more than one workers finish the same task, all of them will be rewarded. A worker may deviate from the system by delaying execution of a task or not executing a task in his task sequence.*

We extend the above example system with probability. In addition to the task sequence, each worker is assigned a probability 0.4 to take the next task, and probability 0.6 to skip the next task. For example, a work will toss a coin before taking the next task: if it is head, he takes the next task, if it is tail, he skip the next task.

Similar to the formalization in [35], in the example, a worker's states are modelled as $X_i = Y_i \times Z_i$, where Y_i is the worker's task sequence and Z_i is the status of each task, $Z_i = \{0, 1, 2\}$, with 0 denoting that the worker is not working, 1 denoting that the work finishes his assigned work, and 2 denoting that the current (finished) work is used by a job. Note that in this example, the

states for altruistic, rational or Byzantine players/workers are exactly the same. A player's initial state is $I_i = (\tau(i,0),0)$. Since the only deviation is not taking the task, the set of actions for all players are the same, i.e., $A_i = \{a_0^i, a_1^i\}$, where a_0^i denotes the player i does not take the task, and a_1^i denotes that the player i will take the task. The action a_1 will lead a state $(\tau(i,j),0)$ to $(\tau(i,j),1)$, and the action a_0 leads to a self-loop. The transition from $(\tau(i,j),1)$ to $(\tau(i,j),2)$ happens if the task $\tau(i,j)$ is used for a job. Note that it does not needs actions from players. Hence, $S = \langle (Y_1,Z_1),\cdots,(Y_n,Z_n)\rangle, I = \langle(\tau(1,0),0),\cdots,(\tau(n,0),0)\rangle$ and $A = \{\{a_0^1,a_1^1\},\cdots,\{a_0^n,a_1^n\}\}$.

We use the function π_i to project a global state s or a global action a to the local state of player i, and use a_τ^i to denote the empty action of player i. The function $B_i(s,a,s')$ is defined as follows:

$$B_i(s,a_i,s') = \begin{cases} \text{true} & \text{if } \pi_i(s) = (\tau(i,j),0), \pi_i(s') = (\tau(i,j),0), a_i = a_0^i \\ \text{true} & \text{if } \pi_i(s) = (\tau(i,j),0), \pi_i(s') = (\tau(i,j),1), a_i = a_1^i \\ \text{false} & \text{otherwise} \end{cases}$$

The first case says that if a player i's next job is j, he can choose to not take the task ($a_i = a_0^i$), and this action leads to a state where i's local state does not change ($\pi_i(s') = \pi_i(s)$). The second case models that if the player takes the task ($a_i = a_1^i$), the action leads to a state where the local state of i changes into $\tau(i,j),1)$.

Differing from the definition in [35], the function $T_i(s,a_i)$ in this system is defined exactly the same as $B_i(s,a_i)$, since both of the actions a_0^i and a_1^i are enabled in the probabilistic system.

$$T_i(s,a_i) = \begin{cases} \text{true} & \text{if } \pi_i(s) = (\tau(i,j),0), a_i = a_0^i \\ \text{true} & \text{if } \pi_i(s) = (\tau(i,j),0), a_i = a_1^i \\ \text{false} & \text{otherwise} \end{cases}$$

The payoff function is defined the same as original as follows.

$$h_i^\psi(s_i,a_i) = \begin{cases} \text{-1} & \pi_i(s) = (\tau(i,j),0) \wedge (a_i = a_1^i) \\ 4 & \pi_i(s) = (\tau(i,j),2) \\ 0 & \text{otherwise} \end{cases}$$

The first case represents that the player works for a task, meaning that the player is paying, instead of gaining, and thus the payoff is negative. The second case capture the situation that the player's task is used in a job and thus the player is paid with positive payoff value. If the player does not work then he gains nothing and pays nothing, and thus has payoff value 0.

The probabilistic function $P_i(s,a_i)$ is defined for altruistic player as follows:

$$P_i(s_i,a_i) = \begin{cases} 0.6 & \text{if } a_i = a_0^i \\ 0.4 & \text{if } a_i = a_1^i \end{cases}$$

If a player is a Byzantine player $i \in Z$, then the probability can be an arbitrary distribution (p can be any value between and include 0 and 1). If a player is an altruistic player $i \notin Z$, the probability distribution is pre-defined by the system (0.6 for not taking the task and 0.4 for taking the task).

7. VERIFICATION ALGORITHM

We are going to verify the Nash Equilibrium of synchronous parallel finite state mechanisms. In order to define the equilibrium we should define a utility function with respect to the execution of the protocol. The concepts of $path$ and $strategy$ has to be used in order to define utility.

The dynamic programming algorithm to evaluate the value and the guaranteed outcome at horizon k for player i from a global state s is as follows.

$v_i^0(Z,s) = u_i^0(Z,s) = 0$ [35]

$$v_i^{k+1}(Z,s) = \begin{cases} i \in Z \\ max_{a_i \in A_i(s)}\{min_{a_{Z-\{i\}} \in A_{Z-\{i\}}}\{ \\ E_{a_{[n]-Z} \in A_{[n]-Z}(s)}(h_i(s,\langle a_{z-\{i\}}, a_i, a_{[n]-Z}\rangle) + \\ \beta_i {v_i}^k(Z,s') | BT(Z,s,\langle a_{-i}, a_i\rangle, s'))\}\} \\ i \notin Z \\ E_{a_i \in A_i(s)}(min_{a_Z \in A_Z(s)}\{ \\ E_{a_{[n]-Z-\{i\}} \in A_{a_{[n]-Z-\{i\}}}}(h_i(s,\langle a_{[n]-Z-\{i\}}, a_Z, a_i\rangle) + \\ \beta_i {v_i}^k(Z,s') | BT(Z,s,\langle a_{-i}, a_i\rangle, s'))\}) \end{cases} \quad (1)$$

$$u_i^{k+1}(Z,s) = \begin{cases} i \in Z \\ min_{a_i \in A_i(s)}\{min_{a_{Z-\{i\}} \in A_{Z-\{i\}}}\{ \\ E_{a_{[n]-Z} \in A_{[n]-Z}(s)}(h_i(s,\langle a_{z-\{i\}}, a_i, a_{[n]-Z}\rangle) + \\ \beta_i {u_i}^k(Z,s') | BT(Z,s,\langle a_{-i}, a_i\rangle, s'))\}\} \\ i \notin Z \\ E_{a_i \in A_i(s)}(min_{a_Z \in A_Z(s)}\{ \\ E_{a_{[n]-Z-\{i\}} \in A_{a_{[n]-Z-\{i\}}}}(h_i(s,\langle a_{[n]-Z-\{i\}}, a_Z, a_i\rangle) + \\ \beta_i {u_i}^k(Z,s') | BT(Z,s,\langle a_{-i}, a_i\rangle, s'))\}) \end{cases} \quad (2)$$

In the probabilistic setting, value functions for individual paths become value functions for path tree from some state. $v_i(\{\pi\})$ and $v_i(\{\pi|k\})$ represent value of the player i in path tree $\{\pi\}$ and the value of length k horizon of path tree $\{\pi\}$

To determine $v_i^k(Z,s)$ at a global state s, we should consider all the globally valid local actions for player i. For each of these actions, Byzantine players can choose a combination of actions which minimize the expected pay-off for player i. Player i can calculate the maxi-min value to choose his rational action. Similarly, for $u_i^k(Z,s)$ player i can calculate the minimin value which defines his least profitable move by playing honestly. We can see the proofs for correctness of this dynamic programming definition later.

Now, we are ready to define $\epsilon - \chi$ Nash Equilibrium.

DEFINITION 2. *Let (S,I,B,T,h,P,β) is a mechanism ($\mathcal{M}$) and $\chi \in \{0,1,...,n\}$ and $\epsilon > 0$. ($\mathcal{M}$) is a $\epsilon - \chi$ Nash-Equilibrium for player $i \in [n]$*

$$\text{if } \forall Z \in \mathcal{P}_\chi([n]-\{i\}), \forall s \in \mathcal{I}_i, \\ u_i(Z,s) + \epsilon \geq v_i(Z \cup \{i\}, s).$$

[35]

The meaning of the above definition is that player i has a Nash Equilibrium if the worst case value (when i is altruistic) obtained among the infinite length paths starting from initial states is at most less than ϵ to the optimal value obtained by i, playing rationally.

It is evident that we need to calculate an infinite sum to evaluate the $\epsilon - \chi$ Nash equilibrium. It would be very useful if we can decide the existence of Nash Equilibrium by looking at sequences of sufficient length. [35] proposes an algorithm to find $\epsilon - \chi$ Nash Equilibrium with an arbitrary precision (δ). We extend their algorithm such that it supports probabilistic systems. Following theorem shows the ability of verifying whether a mechanism is an $\epsilon - \chi$

Nash equilibrium. Initially, it calculates the sufficient path length (k_i), based on the required precision δ, discount factor β and maximum absolute pay-off M. It then calculates $\Delta_i(k_i)$ which is the maximum regret value of being rational after k_i steps. Based on the value of $\Delta_i(k_i)$ we can decide on a range for the actual maximum regret value. If the range fits the precision δ we can decide that the mechanism has $\epsilon + \delta - \chi$ Nash equilibrium.

THEOREM 1. *Let $\mathbb{M} = (S, I, A, B, T, h, P, \beta)$ be an n player mechanism, $\chi \in \{0, \cdots, n\}, \epsilon > 0$ and $\delta > 0$. for each agent $i \in [n]$ define,*

1. $M_i = max\{|h_i(s,a)||s \in S \text{ and } a \in A\}$.
2. $e_i(k) = \beta_i^k \frac{M_i}{1-\beta_i}$.
3. $\Delta_i(k) = max\{v_i^k(Z \cup \{i\}, s) - u_i^k(Z,s)|s \in I, Z \in \mathbb{P}_\chi([n] - \{i\})\}$
4. $\epsilon_1(i,k) = \Delta_i(k) - 10e_i(k)$
5. $\epsilon_2(i,k) = \Delta_i(k) + 10e_i(k)$

For each agent i, Define k_i to be $4E_i(k_i) < \delta$

1. *if $\forall i \in [n]$, $\epsilon \geq \epsilon_2(i,k_i) > 0$ then M is $\epsilon - \chi - Nash$*
2. *if $\exists i \in [n]$ s.t., $0 < \epsilon \leq \epsilon_1(i,k_i)$ then M is not $\epsilon - \chi - Nash$ it can never be a $0 - \chi - Nash$.*
3. *if $\forall i \in [n]$, $\epsilon_1(i,k_i) < \epsilon$ and $\exists j \in [n]$ s.t. $\epsilon < \epsilon_2(j,k_j)$ then M is $(\epsilon + \delta) - \chi - Nash$.*

Though the proofs are presented in appendix, we can provide some intuitions for the proof here. We consider path trees in the probabilistic case instead of paths used in non-probabilistic case. For the definition of $\epsilon - \chi$ Nash equilibrium we need to determine a possible range for $v_i - u_i$ by looking at finite length execution paths trees of length k. For a given length k, we will be able to prove that $v_i - u_i$ lies between $\Delta_i(k) - 10e_i(k)$ and $\Delta_i(k) + 10e_i(k)$ where $\Delta_i(k)$ is the maximum regret value for player i in k-length executions. We can see that if ϵ is greater than the maximum possible value for $v_i - u_i$, $\Delta_i(k) + 10e_i(k)\ \forall i$ the mechanism is obviously an $\epsilon - \chi$ Nash Equilibrium. If for some i, ϵ is less than the minimum possible value for $v_i - u_i$, $\Delta_i(k) - 10e_i(k)$ the mechanism is obviously not an $\epsilon - \chi$ Nash Equilibrium. If ϵ lies in the possible range of $v_i - u_i$, for some i and ϵ is greater than the range for other i's we cannot guarantee about $\epsilon - \chi$ Nash Equilibrium. We surely not that, if the range cannot be bound by ϵ it can be bounded by something larger than ϵ. According to the choice of k, we can prove that, if ϵ is in the range, whole range is guaranteed to be bound by $\epsilon + \delta$ (δ is an upper bound for the size of the uncertainty range of $v_i - u_i$).

Now we can look into how the value of the uncertainty range is evaluated. First we evaluate $e_i(k)$ which is an upper bound for the absolute value of the truncation error introduced by considering the k-length path tree. For the rest of the proof, we use three kinds of path tree pay-offs namely, $v_i(\pi)$, $v_i^k(\hat{\pi}_k)$ and $v_i^k(\pi|k)$ which correspond to pay-offs of infinite optimal path tree, k-optimal path tree and k-prefix of infinite optimal path tree respectively. For a given strategy σ, the absolute difference between payoffs for k-prefix of infinite optimal path tree and k-optimal path tree can be bounded by $2e_i(k)$. Similarly, absolute difference between payoffs for k-optimal path tree and infinite optimal path tree can be bounded by $3e_i(k)$. This proof follows from the triangle inequality which breaks down the said value to a summation of of two terms which match previously proven bounds. Final lemma determines the bound for the absolute difference between k-optimal and infinite optimal for the infinite optimal strategy and k-optimal strategy by a player. This lemma also follows from triangle inequality. Our final bound for absolute difference is $5e_i(k)$. This bound is for either v_i or u_i. But, we need to calculate the error $|(v_i - u_i) - (v_i^k - u_i^k)|$. By rearranging terms we arrive at, $|(v_i - v_i^k) - (u_i - u_i^k)|$ and triangle inequality, introduces the bound as $10e_i(k) = 5e_i(k) + 5e_i(k)$. Since the absolute difference is bounded by $10e_i(k)$, actual difference has a range of size $20e_i(k)$ which happens to be the guaranteed minimum shift to determine a bound for maximum actual regret value.

Algorithm 2 $CheckNash$(mechanism $\mathcal{M}$, int χ, double ϵ , δ)

1: **for all** $i \in [n]$ **do**
2: Let k, $20e_i(k) \leq \delta$
3: Let $s \in S$ and $Z \in \mathcal{P}([n] - \{i\})$
4: $v_i^0(Z,s) \leftarrow 0; u_i^0(Z,s) \leftarrow 0;$
5: **for** t=1 to k **do**
6: $v_i^{t+1}(Z \cup \{i\}, s) \leftarrow \{max_{a_i \in A_i(s)} E_{a_{[n]-Z-\{i\}} \in A_{[n]-Z-\{i\}}(s)} min_{a_Z \in A_Z} \{h_i(s, \langle a_Z, a_{[n]-Z-\{i\}}, a_i\rangle) + \beta_i^t {v_i}^t(Z,s')|BT(Z,s,\langle a_{-i}, a_i\rangle, s')\}\}$
7: $u_i^{t+1}(Z,s) \leftarrow \{E_{a_i \in A_i(s)} min_{a_Z \in A_Z(s)} \{h_i(s, \langle a_{[n]-Z-\{i\}}, a_Z, a_i\rangle) + E_{a_{[n]-Z-\{i\}} \in A_{[n]-Z-\{i\}}} \beta_i^t {u_i}^t(Z,s')|BT(Z,s,\langle a_{-i}, a_i\rangle, s')\}\}$
8: **end for**
9: $\Delta_i \leftarrow \max\{v_i^k(Z \cup \{i\}, s) - u_i^k(Z,s)|s \in I, Z \in \mathcal{P}_\chi([n]\ \{i\})\}$
10: $\epsilon_1(i) \leftarrow \Delta_i - 10e_i(k)$; $\epsilon_2(i) \leftarrow \Delta_i + 10e_i(k)$
11: **if** $\epsilon < \epsilon_1(i)$ **then**
12: **return** FAIL
13: **end if**
14: **end for**
15: **if** $\forall i \in [n]$ $(\epsilon_2(i) < \epsilon)$ **then**
16: **return** PASS with ϵ
17: **else**
18: **return** PASS with $(\epsilon + \delta)$
19: **end if**

Explanation of the Algorithm.

Firstly, we should know how Byzantine players are treated in the algorithm. As stated in section 6 Byzantine players are also treated as rational players for the calculation of guaranteed values for Altruistic players. So the actions of the Byzantine players are always denoted as non deterministic while Altruistic players always have deterministic probabilistic actions. As a total system, the actions are non-deterministic in which the non-determinism is induced totally by Byzantine players.

First 4 lines work on initialization of the algorithm for player i. It defines value k which is the sufficient length for analysis by using δ and $e_i(k)$. $e_i(k)$ is defined in terms of maximum absolute pay-off (M) and discount factor (β). It also initializes the set of Byzantine players and the expected value (guaranteed and optimal) for player i, at any state after 0 steps starting from that state. Line 6 and 7 iteratively calculates expected values for t iterations starting from each state. After the loop execution we have successfully calculated expected values for k length paths starting from any state. Line 9 calculates the maximum difference between expected optimal value

and expected guaranteed value, for k length paths, starting from any initial state, for player i. Line 10 defines a range for existence of actual incentive difference. Line 11 says if the needed incentive difference ϵ is not in the range return $FAIL$. Line 15 to 18 follows from the result of Theorem 1.

Correctness proof of the algorithm shows the correctness of dynamic programming definition and applies the result of Theorem 1 for Nash equilibrium verification.

Correctness of the dynamic programming definition.

Since we have probabilistic strategies we do not always consider paths. We consider a path tree. We optimize expected value of the paths in the path tree. The original pay-off function for a horizon t is

Let π'^a be a path of length k.

$$v_i^{'k}(Z,s) = \begin{cases} k > 0 \wedge i \in Z \\ max_{\pi'^a_i \in \Pi_{t=0}^{k-1} A_i} min_{\pi'^a_{[Z]-\{i\}} \in \Pi_{t=0}^{k-1} A_{[Z]-\{i\}}} \{\{E_{\pi'^a_{[n]-Z} \in \Pi_{t=0}^{k-1} A_{[n]-Z}} (\Sigma_{t=0}^{k} \beta_i^t h_i(\pi'^s(t), \pi'^a(t)))| \\ BT(Z, \pi'^s(t), \pi'^a(t), \pi'^s(t+1)) \wedge \pi'^s(0) = s \wedge |\pi'| = k\}\} \\ k > 0 \wedge i \notin Z \\ E_{\pi'^a_i \in \Pi_{t=0}^{k-1} A_i} (min_{\pi'^a_Z \in \Pi_{t=0}^{k-1} A_Z} \{E_{\pi'^a_{[n]-Z-\{i\}} \in \Pi_{t=0}^{k-1} A_{[n]-Z-\{i\}}} (\Sigma_{t=0}^{k} \beta_i^t h_i(\pi'^s(t), \pi'^a(t)))| \\ BT(Z, \pi'^s(t), \pi'^a(t), \pi'^s(t+1)) \wedge \pi'^s(0) = s \wedge |\pi'| = k\}) \\ k = 0 \end{cases} \tag{3}$$

We can use induction to prove that dynamic programming definition of $v_i^k(Z,s)$ also computes the same function. Induction is done on the path length l.[35]

THEOREM 2. $v_i^{'k}(Z,s) = v_i^k(Z,s)\ \forall k \geq 0$.

PROOF. For $k = 0$ case the result is trivial because value of empty path is defined to be 0. So the maximin path value is 0. Substitute the original function value ($v_i^{'k}(Z,s)$) to compute $v_i^{k+1}(Z,s)$ in the dynamic programming definition. Let $i \in Z$, any path of length k th iteration be π and let $\{\pi\}$ be any infinite length probabilistic path starting from state s.

$$\begin{aligned} & E_{a_{[n]-Z} \in A_{[n]-Z}} (h(s, \langle a_i, a_{-i}) + \beta_i v_i^{'}(Z,s')|BT(Z,s,\langle a_i, a_{-i}\rangle, s')) \\ = & E_{a_{[n]-Z} \in A_{[n]-Z}} (h(s, \langle a_i, a_{-i}\rangle) + \beta_i E_{\pi^a \in \Pi_{t=0}^{k-1} A_{[n]-Z}} (\Sigma_{t=0}^{k-1} \beta_i^t h_i(\pi'^s(t), \pi'^a(t))|BT(Z,s,\langle a_i, a_{-i}\rangle, s'))) \\ = & E_{a_{[n]-Z} \in A_{[n]-Z}} (h(s, \langle a_i, a_{-i}\rangle)) + E_{A_{[n]-Z}} (E_{\pi'^a \in \Pi_{t=1}^{k} A_{[n]-Z}} (\Sigma_{t=1}^{k} \beta_i^t h_i(\pi'^s(t), \pi'^a(t))|BT(Z,s,\langle a_i, a_{-i}\rangle, s'))) \\ = & E_{\pi'^a \in \Pi_{t=0}^{k} A_{[n]-Z}} (h(s, \langle a_i, a_{-i}\rangle)) + E_{\pi'^a \in \Pi_{t=0}^{k} A_{[n]-Z}} (\Sigma_{t=1}^{k} \beta_i^t h_i(\pi'^s(t), \pi'^a(t))|BT(Z,s,\langle a_i, a_{-i}\rangle, s')) \\ = & E_{\pi'^a \in \Pi_{t=0}^{k} A_{[n]-Z}} (\Sigma_{t=0}^{k} \beta_i^t h_i(\pi'^s(t), \pi'^a(t))|BT(Z,s,\langle a_i, a_{-i}\rangle, s')) \end{aligned}$$

maximin value would be

$$v_i^{'k}(Z,s) = max_{\pi'^a_i \in \Pi_{t=0}^{k} A_i} min_{\pi'^a_{[Z]-\{i\}} \in \Pi_{t=0}^{k} A_{[Z]-\{i\}}} E_{\pi'^a \in \Pi_{t=0}^{k} A_{[n]-Z}} (\Sigma_{t=0}^{k} \beta_i^t h_i(\pi'^s(t), \pi'^a(t))|BT(Z,s,\langle a_i, a_{-i}\rangle, s')) \tag{4}$$

When $\{\pi\}$ is the path set corresponding to optimal expected value for length k, $\{\pi'\}$ would be the path set $s\langle a_i, a_{-i}\rangle \pi \in \Pi_{t=0}^{k} A_{[n]-Z}$. Suppose we choose a different path set from s' other than $v_i^k(s', Z)$ if does not correspond to the maximum expected value over player i among the guaranteed values, we can miss a max value path set in general. □

Proof is similar for $u_i^k(Z,s)$.

8. CASE STUDY: ANALYZING A PROBABILISTIC SECRET SHARING SCHEME

we performed a case study on a probabilistic secret sharing scheme [22] by modelling according to the proposed probabilistic approach in section **??** with some protocol specific improvements.

Secret sharing is a way for a group of users to share a *secret*: the secret is distributed among the group members, such that each member has a *share* of the secret; and the secret can only be reconstructed with a sufficient number of shares. In other words, the secret is kept confidential, even in the presence of a limited number of traitors (compromised members or non-cooperating members who prevent the reconstruction of the secret). This property makes secret sharing an ideal scheme for storing high sensitive informations, such as encryption keys and launch codes. In the parametric form, a n-m ($n \geq m$) secret sharing ensures that it requires at least m honest players to reconstruct the secret within a group of n players. In other words, less than m compromised players together cannot reconstruct the secret; and less than $n - m$ non-cooperating members together cannot stop the honest players to reconstruct of the secret.

The most well-known secret sharing scheme is the Shamir's secret sharing [39]. However the scheme does not work when the players are *rational* – a player follows the protocol only if it increases his utility. It is shown that a rational player will not share his secret with anyone else. [22]. In fact, it is proved that in general, in all practical mechanisms for shared-secret reconstruction with an upper bound on the running time, the best strategy for each rational player is doing nothing [22]. Hence, a randomized mechanism (the probabilistic secret sharing scheme) is proposed to ensure that rational players would reveal their shares to reconstruct the secret, i.e., achieving a Nash-Equilibrium with a constant expected running time.

The probabilistic scheme is a 3-3 secret sharing[2]. Given the 3 players, we index each player is indexed with a distinct number in $\{1,2,3\}$. We use i to denote one player, then the other two players are $(i+1)\%3$, denoted as i^+, and $(i-1)\%3$, denoted as i^-. There is an issuer who assign each player a share of secret. Assuming the share cannot be changed; for example, it is signed by the issuer. The scheme works as follows.

1. Each player i
   - chooses a bit c_i (c_i can only be 1 or 0) such that $c_i = 1$ with probability α and $c_i = 0$ with probability $1 - \alpha$;
   - chooses a bit $c_{(i,+)}$ ($c_{(i,+)}$ can only be 1 or 0) randomly so that $c_{(i,+)} = 1$ with probability $1/2$ and $c_{(i,+)} = 0$ with probability $1/2$;
   - calculates $c_{(i,-)} = c_i \oplus c_{(i,+)}$.

   Then player i sends the bit $c_{(i,+)}$ to player i^+ and sends the bit $c_{(i,-)}$ to player i^-. This means that the player i receives a bit $c_{(i^+,-)}$ from player i^+ and a bit $c_{(i^-,+)}$ from player i^-.
2. On receiving $c_{(i^+,-)}$, player i computes $c_{(i^+,-)} \oplus c_i$ and sends it to player i^-. Correspondingly, i receives $c_{((i^+)^+,-)} \oplus c_{i^+}$ from i^+. Since there are only 3 players, player $(i^+)^+$ is the player i^-. Hence $c_{((i^+)^+,-)} \oplus c_{i^+} = c_{(i^-,-)} \oplus c_{i^+}$.
3. When receiving both $c_{(i^-,+)}$ from player i^- (step 1) and receiving $c_{(i^-,-)} \oplus c_{i^+}$ from i^+ (step 2), i computes $p =$

[2]A general n-m secret sharing is considered as variations of the basic 3-3 secret sharing (for details, see [22])

$c_{(i^-,+)} \oplus (c_{(i^-,-)} \oplus c_{i^+}) \oplus c_i$. Note that $c_{(i^-,+)} \oplus (c_{(i^-,-)} \oplus c_{i^+}) \oplus c_i = c_{i^-} \oplus c_{i^+} \oplus c_i$, i.e., $p = c_1 \oplus c_2 \oplus c_3$ (It holds for all players). If $p = c_i = 1$ then player i sends his share to the others.

4. Depending on the value of p and c_i, player i decides whether to send his share.

   - If $p = 1$, i receives 3 shares, then he construct the secret.
   - If $p = 0$ and i received no secret shares, or if $p = 1$ and i received exactly one share (possibly from itself; that is, we allow the case that i did not receive any shares from other players but sent its own), the issuer restarts the protocol.
   - Otherwise, player i stops the protocol. Because someone must have been cheating.

   The intuition behind the last step is as follows: When $p = 0$, no one should send his share; and thus i receives no secret share. Otherwise, some one calculated wrong (cheating). When $p = 1$ either $c_i = c_{i^+} = c_{i^-} = 1$ or there exists exactly one player chooses 1, i.e., $c_i = c_{i^+} = 0$ and $c_{i^-} = 1$, $c_i = c_{i^-} = 0$ and $c_{i^+} = 1$, or $c_{i^+} = c_{i^-} = 0$ and $c_i = 1$. In the former case, each player can construct the secret. In the later case, i should receive exactly one share. Otherwise, some player calculated wrong (cheating).

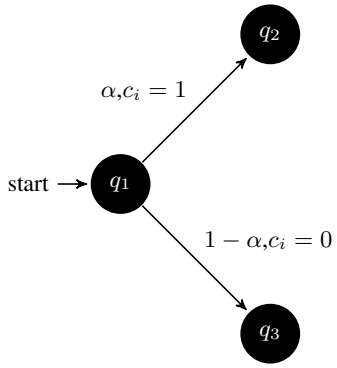
Model.

The above randomized version of the shamir secret sharing scheme can be modelled with the probabilistic extension proposed.

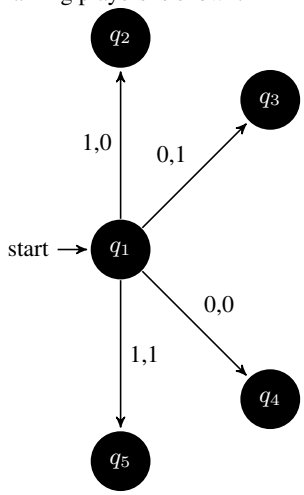
Here, we assume that message channels are secure and sender's authenticity is preserved. The above protocol is a deterministic protocol where at each step an honest player has to perform a single predefined task(may be probabilistic), depending on the global state he/she is in. This scenario has a unique initial global state. Let's say player i's initial state is s_{i0}. The initial global state is is unique here so that, $I = \{(s_{10}, s_{20}, s_{30})\}$. Let's have a look at their Byzantine behaviours starting from the initial state.

1. Player i's probabilistic action to choose c_i. $c_i = 1$ with probability α.
2. choose c_{i^+} with 0.5 probability.
3. send messages to (i^+ and i^-. (since the messages are boolean, player has four different actions)
4. send 'xor' to player i^-. (two options - sending correct one or incorrect one)
5. send or not send the messages to player i^- and i^+. (total four possibilities)
6. restart the protocol

The above actions sets are activated in different levels. For example, first action set starts from the initial state.



Second action is, sending two other players two random values calculated. For a Byzantine player there are four possibilities. In the diagram possibilities for sending different binary values to two remaining players is shown.



In this example, the set $Z = \{\}$ so that each player is either Altruistic or Rational. The pay-off function is only defined for actions which end in terminal states. For all other actions pay-off is simply 0. Let the value $\beta_i = 1\ \forall i$ (which is a deviation from the proposed approach) For any infinite length path, the value function would always be a finite value. As a result, discount factor is not essential here. Possible values of the value functions according to these reductions would be an expected utility value under some non-deterministic choice. Then the probabilistic path for maximum utility is same as the path for maximum value in this model. Terminal actions are the actions where at least one player learns the secret. In all the other actions, the transition state would be the initial state itself.
The protocol was modelled in PAT to verify the reaching probabilities of each terminal state. It was verified that the protocol is a Nash equilibrium when $\alpha_i = 0.3$ for each i.

9. EXPERIMENTAL RESULTS

Experiments are based on the example mechanism presented in section 1. Exact same mechanism is tested for running time and memory performance for different numbers of players and different Byzantine set sizes. In all cases $\epsilon = 0.01$ and $\delta = 0.005$. Experiments were run in 64-bit x-86 Intel machine having 2.40 GHz processor with 8GB RAM.

Table 1: Experimental Results

Agents	Jobs	Byzantines	Nash	CPU(sec)	Present State Bits	Action Bits
3	2	1	PASS	7.8405	9	3
3	2	2	PASS	16.7238	9	3
3	2	3	FAIL	7.6124	9	3
4	2	1	PASS	979.3278	12	4
4	2	2	FAIL	1667.687	12	4
4	2	3	FAIL	1892517	12	4

10. CONCLUSIONS AND FUTURE WORK

We presented a model checking algorithm to verify Nash Equilibrium in deterministic, probabilistic protocols in finite state concurrent systems. We specify such protocols as probabilistic finite automata and verify $\epsilon - f - Nash$[35] by using the proposed algorithm. We evaluated the applicability of the protocol with a case study about randomized secret sharing protocol[22] and presented the experimental results via a continuous task sharing scenario.

As future work, we will support non-deterministic protocols with the presence of probabilistic actions (Markov Decision Processes). We also plan to develop a tool to convert a finite state concurrent system specification to a game with the least involvement of the designer.

11. REFERENCES

[1] Analysis of probabilistic contract signing. (October):1–26, 2006.

[2] I. Abraham, L. Alvisi, and J. Y. Halpern. Distributed computing meets game theory: combining insights from two fields. *SIGACT News*, 42(2):69–76, 2011.

[3] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial-order reduction in symbolic state space exploration. In *Computer Aided Verification*, pages 340–351. Springer, 1997.

[4] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. MOCHA: modularity in model checking. In *Proc. 10th Computer Aided Verification*, volume 1427 of *LNCS*, pages 521–525. Springer, 1998.

[5] M. Backes, O. Ciobotaru, and A. Krohmer. Ratfish: A file sharing protocol provably secure against rational users. In *Proc. 15th European Symposium on Research in Computer Security*, volume 6345 of *LNCS*, pages 607–625. Springer, 2010.

[6] C. Baier, J.-P. Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[7] P. Ballarini, M. Fisher, and M. Wooldridge. Automated game analysis via probabilistic model checking: a case study. *Electr. Notes Theor. Comput. Sci.*, 149(2):125–137, 2006.

[8] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[9] P. Bouyer, R. Brenguier, N. Markey, and M. Ummels. Concurrent games with ordered objectives. In *Foundations of software science and computational structures*, pages 301–315. Springer, 2012.

[10] R. Brenguier. PRALINE: A tool for computing nash equilibria in concurrent games. In *Proc. 25th Computer Aided Verification*, volume 8044 of *LNCS*, pages 890–895. Springer, 2013.

[11] L. Buttyan and J.-P. Hubaux. Toward a {F}ormal {M}odel of {F}air {E}xchange - a {G}ame {T}heoretic {A}pproach. (December 1999):1–16, 1999.

[12] T. Chen, V. Forejt, M. Z. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[13] X. Chen and X. Deng. Settling the complexity of 2-player nash-equilibrium. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 2006.

[14] C. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. GAVS+: an open platform for the research of algorithmic game solving. In *Proc. 17th Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 258–261. Springer, 2011.

[15] C. Daskalakis. On the complexity of approximating a nash equilibrium. *ACM Transactions on Algorithms*, 9(3):23, 2013.

[16] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *Communications of the ACM*, 52(2):89, 2009.

[17] C. Daskalakis, A. Mehta, and C. H. Papadimitriou. Progress in approximate nash equilibria. In *Proc. 8th ACM Conference on Electronic Commerce*, pages 355–358. ACM, 2007.

[18] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *J. Comput. Syst. Sci.*, 68(2):374–397, 2004.

[19] A. Fabrikant, C. H. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *Proc. 36th Annual ACM Symposium on Theory of Computing*, pages 604–612. ACM, 2004.

[20] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

[21] V. Gripon and O. Serre. Qualitative concurrent stochastic games with imperfect information. In *Automata, Languages and Programming*, pages 200–211. Springer, 2009.

[22] J. Halpern and V. Teague. Rational secret sharing and multiparty computation: Extended abstract. In *Proc. 36th Annual ACM Symposium on Theory of Computing*, pages 623–632. ACM, 2004.

[23] J. Hao, S. Song, Y. Liu, J. Sun, L. Gui, J. S. Dong, and H.-f. Leung. Probabilistic model checking multi-agent behaviors in dispersion games using counter abstraction. In *PRIMA 2012: Principles and Practice of Multi-Agent Systems*, pages 16–30. Springer, 2012.

[24] T. A. Henzinger. Model checking game properties of multi-agent systems (abstract). In *Proc. 25th Automata, Languages and Programming*, volume 1443 of *LNCS*, page 543. Springer, 1998.

[25] R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, S. Pandav, A. Slobodová, C. Taylor, V. Frolov, E. Reeber, et al. Replacing testing with formal verification in intelˆ{\ scriptsize\ circledR} coretm i7 processor execution engine validation. In *Computer Aided Verification*, pages 414–429. Springer, 2009.

[26] S. Kremer and J. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *Journal of Computer Security*, 11(3):399–430, 2003.

[27] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. *CONCUR 2001âĂŤConcurrency Theory*, pages 551–565, 2001.

[28] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification*,

CAV'11, pages 585–591, 2011.

[29] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proc. 7th Symposium on Operating Systems Design and Implementation*, pages 191–204. USENIX Association, 2006.

[30] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard. A cooperative internet backup scheme. In *Proc. the General Track: 2003 USENIX Annual Technical Conference*, pages 29–41. USENIX, 2003.

[31] A. Lomuscio, H. Qu, and F. Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. 21st Computer Aided Verification*, volume 5643 of *LNCS*, pages 682–688. Springer, 2009.

[32] A. Lomuscio and F. Raimondi. Model checking knowledge, strategies, and games in multi-agent systems. In *Proc. 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 161–168. ACM, 2006.

[33] A. Mahimkar and V. Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *Proc. 18th IEEE Computer Security Foundations Workshop*, pages 287–301. IEEE Computer Society, 2005.

[34] F. Mari. Verification and synthesis for discrete time linear hybrid systems.

[35] F. Mari, I. Melatti, I. Salvo, E. Tronci, L. Alvisi, A. Clement, and H. C. Li. Model checking nash equilibria in MAD distributed systems. In *Formal Methods in Computer-Aided Design*, pages 1–8. IEEE, 2008.

[36] M. J. Osborne and A. Rubinstein. *A course in game theory*. MIT press, 1994.

[37] P. project. Pat: Process analysis toolkit. pat.comp.nus.edu.sg, last visted at 25 Feb. 2016.

[38] M. O. Rabin. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267, 1983.

[39] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[40] J. Shneidman and D. C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing*, pages 88–97. ACM, 2004.

[41] A. Toumi, J. Gutierrez, and M. Wooldridge. A tool for the automated verification of nash equilibria in concurrent games. In *Proc. 12th Theoretical Aspects of Computing*, volume 9399 of *LNCS*, pages 583–594. Springer, 2015.

[42] Y. Zhang, C. Zhang, J. Pang, and S. Mauw. Game-based verification of contract signing protocols with minimal messages. *ISSE*, 8(2):111–124, 2012.

APPENDIX

A. PROOFS

In order to prove the convergence of the value function we should find a function to determine the value difference.
Define $e_i(k) = \beta_i^k \frac{M_i}{1-\beta_i}$.

LEMMA 1. *1.* $\forall k \in \mathbb{N}, |v_i(\{\pi\}) - v_i(\{\pi|_k\})| \leq e_i(k)$.

2. $\lim_{k\to\infty} v_i(\{\pi\}) = v_i(\{\pi|_k\})$

PROOF. Consider the absolute difference for T-length path tree and fixed k-length path tree as follows.

1. $|v_i(\{\pi|_T\}) - v_i(\{\pi|_k\})| =$
$|E_{\pi'^a \in \Pi_{t=0}^{k-1} A_{[n]-Z}}(\Sigma_{t=0}^{k-1}\beta_i^t h_i(\pi^s(t), \pi^a(t))) -$
$E_{\pi''^a \in \Pi_{t=0}^{k-1} A_{[n]-Z} \Pi_{t=k}^{T} A_{[n]-Z}}(\Sigma_{t=0}^{T}\beta_i^t h_i(\pi''^s(t), \pi''^a(t)))|$

We need to equalize the variable set considered expected value. We just append the variables that do not appear in the expected value function to the considered variable set since it does not make any difference.

$|E_{\pi''^a \in \Pi_{t=0}^{k-1} A_{[n]-Z} \Pi_{t=k}^{T} A_{[n]-Z}}(\Sigma_{t=0}^{k-1}\beta_i^t h_i(\pi^s(t), \pi^a(t))) -$
$E_{\pi''^a \in \Pi_{t=0}^{k-1} A_{[n]-Z} \Pi_{t=k}^{T} A_{[n]-Z}}(\Sigma_{t=0}^{T}\beta_i^t h_i(\pi''^s(t), \pi''^a(t)))|$
$|E_{\pi''^a \in \Pi_{t=0}^{k-1} A_{[n]-Z} \Pi_{t=k}^{T} A_{[n]-Z}}(\Sigma_{t=k}^{T}\beta_i^t h_i(\pi''^s(t), \pi''^a(t)))|$
$\leq |\Sigma_{t=k}^{T} E_{\pi''^a \in \Pi_{t=0}^{k-1} A_{[n]-Z} \Pi_{t=k}^{T} A_{[n]-Z}}(\beta_i^t h_i(\pi''^s(t), \pi''^a(t)))|$
(Linearity of Expectation)
$\leq \Sigma_{t=k}^{T} |E_{\pi''^a \in \Pi_{t=0}^{k-1} A_{[n]-Z} \Pi_{t=k}^{T} A_{[n]-Z}}(\beta_i^t h_i(\pi''^s(t), \pi''^a(t)))|$
(Triangle inequality)
$\leq \beta_i^k \frac{M_i}{1-\beta_i}$ (By the choice of M_i and $\lim_{T\to\infty}$)
$\leq e_i(k)$

2. $\lim_{k\to\infty} e_i(k) = 0$,
$\lim_{k\to\infty} |v_i(\{\pi\}) - v_i(\{\pi|_k\})| = 0$ (by comparison test)
$\lim_{k\to\infty} v_i(\{\pi\}) = v_i(\{\pi|_k\})$.

□

LEMMA 2. *Let* $\{\pi\}$ *be a path tree s.t.* $v_i(\{\pi\})$ *is minimum in* $Path(s, Z, i, \{\sigma\})$. *Let* $\{\{\bar{\pi}_k\}\}_{k\in\mathbb{N}}$ *be a sequence of finite path, s.t.* $\forall k$, $v_i(\{\bar{\pi}_k\})$ *is minimum in* $Path_k(s, Z, i, \{\sigma\})$. *Then* $v_i(\{\pi|_k\}) - v_i(\{\bar{\pi}_k\}) \leq 2e_i(k)$

[34]

LEMMA 3. $\forall Z \in [n]$, $\forall$*state* $s \in S$ *and* $\forall$*strategy sets* $\{\sigma\}$ *we have*
$\lim_{k\to\infty} v_i^k(Z, s, \{\sigma\}|_k) = v_i(Z, s, \{\sigma\})$

[34]

LEMMA 4. $\forall Z \subseteq [n]$, $\forall$*states* $s \in S$ *and* $\forall k \in \mathbb{N}$, *we have:*
$|v_i^k(Z, s) - v_i(Z, s)| < 5e_i(k)$

[34]

LEMMA 5. $\forall Z \subseteq [n]$, $\forall$*states s and* $\forall k \in \mathbb{N}$, *we have:*
$|u_i^k(Z, s) - u_i(Z, s)| < 5e_i(k)$

[34]

By referring to the above lemmas we can state the following proposition.

PROPOSITION 1. *1.* $\lim_{k\to\infty} v_i^k(Z, s) = v_i(Z, s)$

2. $\lim_{k\to\infty} u_i^k(Z, s) = u_i(Z, s)$

[34]

Now we can prove the theorem 1.
By Lemma 4 and 5 we have,
$\forall Z \in P_\chi([n] - \{i\})$ and $\forall statess \in S$
$|v_i^k(Z, s) - v_i(Z, s)| < 5e_i(k)$
$|u_i^k(Z, s) - u_i(Z, s)| < 5e_i(k)$
This will imply,
$v_i(Z \cup \{i\}, s) \leq v_i^k(Z \cup \{i\}, s) + 5e_i(k)$ By Lemma 2
$v_i(Z \cup \{i\}, s) \geq v_i^k(Z \cup \{i\}, s) - 5e_i(k)$ By Lemma 3
$u_i(Z, s) \leq u_i^k(Z, s) + 5e_i(k)$ By Lemma 4
$u_i(Z, s) \geq u_i^k(Z, s) - 5e_i(k)$ By Lemma 5
[34] Now, we can prove the 3 statements.

1. Using Lemma 2 and Lemma 5,
   $v_i(Z \cup \{i\}, s) - u_i(Z, s) \leq v_i^k(Z \cup \{i\}, s) + 5e_i(k) - (u_i^k(Z, s) - 5e_i(k))$
   $= v_i^k(Z \cup \{i\}, s) - u_i^k(Z, s) + 10e_i(k)$
   $\leq \Delta_i(k) + 10e_i(k)$
   if $\epsilon \geq \epsilon_2(i, k)$ then $\Delta_i(k) \leq \epsilon - 10e_i(k)$
   So, $\forall Z \in P_\chi([n] - \{i\})$ and $\forall s \in I$,
   $v_i(Z \cup \{i\}, s) - u_i(Z, s) \leq \epsilon$
   M is $\epsilon - \chi - Nash$.
2. Similarly, 3 and 4 can be used to prove
   $v_i(Z \cup \{i\}, s) - u_i(Z, s) \geq v_i^k(Z \cup \{i\}, s) - 5e_i(k) - (u_i^k(Z, s) + 5e_i(k))$
   $= v_i^k(Z \cup \{i\}, s) - u_i^k(Z, s) - 10e_i(k)$
   if $\epsilon \leq \epsilon_1(i, k)$ then $\Delta_i(k) \geq \epsilon + 10e_i(k)$
   This implies $\exists Z \in P_\chi([n] - \{i\})$ and $s \in I$ s.t.
   $v_i(Z \cup \{i\}, s) - u_i(Z, s) \geq \epsilon$
   M is not $\epsilon - \chi - Nash$.
3. if $\forall i$, $\epsilon_1(i, k_i) < \epsilon$ and for some $j, \epsilon < \epsilon_2(j, k_j)$,
   it is not possible to decide whether M is $\epsilon - \chi - Nash$. But, since $\epsilon_2(j, k_j) - \epsilon_1(i, k_i) = 20e_i(k_i)$ and $20e_i(k_i) < \delta$, we have,
   $\forall i \in [n]$, $\epsilon + \delta > \epsilon_2(i, k_i)$. According to statement 1, we have
   $(\epsilon + \delta) - \chi - Nash$.

[34]

B. MODELLING RANDOMIZED SECRET SHARING ACCORDING TO THE SPECIFICATION

The protocol we model has 3 players.

States can be numbered based on the stage of the protocol and the number of actions enabled in previous stage. We can identify stages numbered from 0 to 6.
Define a function size to determine the number of states in a particular stage.

- stage 0 - initial state - size(0)= 1
- stage 1 - after choosing c_i - size(1)= 2
- stage 2 - after choosing $c_{i,i+1}$ - size(2)= 2
- stage 3 - after sending $c_{i,i-1}$ - size(3)= 2
- stage 4 - after sending xor - size(4)= 2
- stage 5 - after sending m to $i+1$ - size(5)= 2
- stage 6 - after sending m to $i-1$ - size(6)= 2

A state can be represented by a sequence of integers with length denoting the stage it belongs to, and each number of the sequence belonging to the chosen action in each of the previous states. This numbering mechanism is similar of all the players.
$S_1 = \{x \in seq(\{1, 2\}) || x| \leq 7 \forall i, x_i \leq size(i-1)\}$
$S_2 = \{x \in seq(\{1, 2\}) || x| \leq 7 \forall i, x_i \leq size(i-1)\}$
$S_3 = \{x \in seq(\{1, 2\}) || x| \leq 7 \forall i, x_i \leq size(i-1)\}$

Actions.

We have 3 action sets, A_1 , A_2 and A_3.
$A_1 = \{c_1, nc_1, c_{12}, nc_{12}, sr_{12}, sr_{13}, sr_{21}, sr_{31}, nsr_{12}, nsr_{13}, nsr_{21}, nsr_{31}, xor_{13}, xor_{21}, nxor_{13}, nxor_{21}, m_{12}, m_{13}, m_{21}, m_{31}, nm_{12}, nm_{13}, nm_{21}, nm_{31}, e\}$
$A_2 = \{c_2, nc_2, c_{23}, nc_{23}, sr_{23}, sr_{21}, sr_{32}, sr_{12}, nsr_{23}, nsr_{21}, nsr_{32}, nsr_{12}, xor_{21}, xor_{32}, nxor_{21}, nxor_{32}, m_{21}, m_{23}, m_{12}, m_{32}, nm_{21}, nm_{23}, nm_{12}, nm_{32}, e\}$
$A_3 = \{c_3, nc_3, c_{31}, nc_{31}, sr_{31}, sr_{32}, sr_{23}, sr_{13}, nsr_{31}, nsr_{32}, nsr_{23}, nsr_{13}, xor_{32}, xor_{13}, nxor_{32}, nxor_{13}, m_{32}, m_{31}, m_{23}, m_{13}, nm_{32}, nm_{31}, nm_{23}, nm_{13}, e\}$

The set of synchronized actions A^s is,
$A^s = \{sr_{12}, sr_{13}, sr_{21}, sr_{23}, sr_{31}, sr_{32}, xor_{13}, xor_{21}, xor_{32} nsr_{12}, nsr_{13}, nsr_{21}, nsr_{23}, nsr_{31}, nsr_{32}, nxor_{13}, nxor_{21}, nxor_{32}\}$

Local Transition function.

Transitions happens between consecutive stages of the protocol. For a transition to be valid, the sequence corresponding to current and next states should vary by exactly 1. Sequence corresponding to the current state should be a prefix of the next state.

Local Action Probabilities.

$$P_1(s_1, a_1) = \begin{cases} \alpha & s_1 = I_1 \wedge a_1 = c_1 \\ 1-\alpha & s_1 = I_1 \wedge a_1 = nc_1 \\ 0.5 & s_1 = I_{11} \vee s_1 = I_{12} \wedge a_1 = c_{12} \vee a_1 = c_{13} \\ 1 & otherwise \end{cases}$$

$$P_2(s_2, a_2) = \begin{cases} \alpha & s_2 = I_2 \wedge a_2 = c_2 \\ 1-\alpha & s_2 = I_2 \wedge a_2 = nc_2 \\ 0.5 & s_2 = I_{21} \vee s_2 = I_{22} \wedge a_2 = c_{21} \vee a_1 = c_{23} \\ 1 & otherwise \end{cases}$$

$$P_3(s_3, a_3) = \begin{cases} \alpha & s_3 = I_3 \wedge a_3 = c_3 \\ 1-\alpha & s_3 = I_3 \wedge a_3 = nc_3 \\ 0.5 & s_3 = I_{31} \vee s_3 = I_{32} \wedge a_3 = c_{32} \vee a_1 = c_{31} \\ 1 & otherwise \end{cases}$$

C. DEFINITIONS OF BASIC CONCEPTS

Here are some general definitions of the concepts used in the paper.

DEFINITION 3. *concurrency : The property of program, algorithm, or problem decomposability into order-independent or partially-ordered components or units.This means that even if the concurrent units of the program, algorithm, or problem are executed out-of-order or in partial order, the final outcome will remain the same. This allows for parallel execution of the concurrent units, which can significantly improve the overall speed of execution in multi-processor and multi-core systems.*

DEFINITION 4. *Protocol : A set of rules that governs the interaction between agents.*

Difference between automata and finite state machine

DEFINITION 5. *Synchronization: Process of precisely coordinating or matching two or more activities, devices, or processes in time*

DEFINITION 6. *Game Theory : Game theory is the mathematical study of interaction among independent, self-interested agents.*

According to the definition of game theory we can define a game.

DEFINITION 7. *Game : Well defined set of interactions among independent self-interested agents.*

DEFINITION 8. *Game: A game is defined as a decision tree (or more precisely, a dag) in which every edge, called move, has a polarity indicating whether it is played by the program, called Proponent, or by the environment, called Opponent.*

DEFINITION 9. *Alternating Game: A play is alternating when Proponent and Opponent alternate strictly - that is, when neither of them plays two moves in a row.*

DEFINITION 10. *Concurrent Game:*

DEFINITION 11. *A (finite) concurrent game is a tuple G=*$\langle h, States, Agt, Act, Mov, Tab\rangle$ *, where* $States$ *is a (finite) set of states,* Agt *is a (finite) set of players,* Act *is a (finite) set of actions, and* $Mov : States \times Agt \rightarrow 2^{act} \setminus \{\phi\}$ *is a mapping indicating the actions available to a given player in a given state;* $Tab : States \times Act^{Agt} \rightarrow States$ *associates with a given state and a given move of the players the resulting state.*

[9] The above definition refers to a global state set and can be used for perfect information games. In imperfect information games, some global states are indistinguishable. Probability can also be in-cooperated in to the state transition function. [21]

DEFINITION 12. *Protocol as a game : A protocol is a set of rules that governs the interaction between agents. When these agents are self-interested, we can model the protocol as a game.*