Model Checking Nash Equilibria in MAD Distributed Systems

Federico Mari, Igor Melatti, Ivano Salvo, Enrico Tronci
Dep. of Computer Science
University of Rome "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
Email: {mari,melatti,salvo,tronci}@di.uniroma1.it

Lorenzo Alvisi, Allen Clement, Harry Li
Dep. of Computer Science
University of Texas at Austin
1 University Station C0500, Austin, Texas, USA
Email: {lorenzo,aclement,harry}@cs.utexas.edu

***Abstract*—We present a symbolic model checking algorithm for verification of Nash equilibria in finite state mechanisms modeling *Multiple Administrative Domains* (MAD) distributed systems.**

Given a finite state mechanism, a *proposed protocol* for each agent and an *indifference threshold* for rewards, our model checker returns *PASS* if the proposed protocol is a Nash equilibrium (up to the given indifference threshold) for the given mechanism, *FAIL* otherwise.

We implemented our model checking algorithm inside the NuSMV model checker and present experimental results showing its effectiveness for moderate size mechanisms.

I. Introduction

Cooperative services are increasingly popular distributed systems in which nodes (agents) belong to *Multiple Administrative Domains* (MAD). Thus in a MAD distributed system each node owns its resources and there is no central authority owning all system nodes. Examples of MAD distributed systems include Internet routing [25], [49], wireless mesh routing [40], file distribution [16], archival storage [41], cooperative backup [6], [17], [37].

In traditional distributed systems, nodes may deviate from their specifications (*Byzantine nodes*) because of bugs, hardware failures, faulty configurations, or even malicious attacks. In MAD systems, nodes may also deviate because their administrators are *rational*, i.e. selfishly intent on maximizing their own benefits from participating in the system (*selfish nodes*). For example, selfish nodes may change arbitrarily their protocol if that is at their advantage.

Cooperative file distribution (e.g. see [16]) is a typical example of the above scenario. Every peer will be happy to download file chunks from other peers. However, in order to save bandwidth, a *selfish* peer may modify its protocol parameters to disallow upload of its file chunks.

In this paper we present an automatic verification algorithm for MAD distributed systems. That is, given a protocol P for a MAD system and a property φ for P we want to automatically verify if φ holds for P.

Note that in a MAD system *any* node may behave selfishly. This rules out the classical approach (e.g. see [48]) of modeling nodes deviating from the given protocol as *Byzantine* [34]. In fact, doing so would leave us with a system in which all nodes are Byzantine. Very few interesting protocols (if any) work correctly under such conditions. Thus, in order to verify MAD systems, we first need a model for them in which protocol correctness can be formally stated and hopefully proved. This issue has been studied in [1], [15] where the *BAR* model has been introduced.

In BAR, a node is either *Byzantine*, *Altruistic*, or *Rational*. Byzantine nodes, as usual, can deviate from their specification in any way for any reason. Altruistic nodes follow their specification faithfully, without considering their self-interest. Rational nodes deviate selfishly from a given protocol if doing so improves their own utility. In the BAR framework correctness of a protocol with respect to a given property is stated as BAR tolerance. Namely, a protocol is *BAR tolerant* if it guarantees the desired property despite the presence of Byzantine and rational players.

Several BAR tolerant protocols have since been proposed [1], [36] to implement cooperative services for p2p backup and live data streaming. Taking into account how hard it is to formally prove correctness for classical distributed protocols it is not surprising that formally proving that a given protocol is BAR tolerant is indeed quite a challenge (e.g. see [15]).

This motivates investigating if the model checking techniques devised for classical distributed protocols can also be used in our framework. To this end we note that in order to show that a protocol is BAR tolerant, it is sufficient to show that it satisfies the given property when all rational nodes follow the protocol *exactly* and then to show that all rational nodes do, in fact, follow the protocol *exactly*.

If all rational nodes follow the given protocol *exactly* we are left with a system with only Byzantine and altruistic nodes. Well known model checking techniques (e.g. see [14] for a survey) are available to verify that such systems satisfy a given property despite the presence of a limited number of Byzantine nodes. It suffices, as usual, to model Byzantine nodes with nondeterministic automata.

Unfortunately, to the best of our knowledge, no model checking algorithm or tool is available to address the second BAR tolerance requirement, namely proving that all rational nodes do follow the given protocol *exactly*.

To fill this gap in this paper we present a symbolic model checking algorithm to automatically verify that it is in the best interest of each rational agent to follow *exactly* the given protocol. This is usually accomplished by proving that no rational agent has an incentive in deviating from the proposed protocol. This, in turn, is done by proving that the proposed protocol is a *Nash equilibrium* (e.g. see [25], [11]).

A. Our contribution

First of all we need a formal definition of *mechanism* suitable for model checking purposes and yet general enough to allow modeling of interesting systems. Accordingly in Sect. III we present a definition of *Finite State Mechanism* suitable for modeling of finite state BAR systems as well as for developing effective verification algorithms for them. We model each agent with a *Finite State Machine* defining its admissible behavior, that is, using the BAR terminology, its *Byzantine* behavior. Each agent action yields a real valued *reward* which depends both on the system state and on agents' actions. The *proposed protocol* constrains which actions should be taken in each state. This protocol defines the behavior of *altruistic* agents.

The second obstruction to overcome is the fact that we need to handle infinite games since nodes are running, as usual, nonterminating protocols. As a result, we need to *rank* infinite sequences of agents' actions (*strategies*). This is done in Sects. V, VI by using a *discount factor* (as usual in game theory [26]) to decrease relevance of rewards *too far* in the future. In Prop. 1 we give a dynamic programming algorithm to effectively compute the value of a finite strategy in our setting.

To complete our framework we need a notion of equilibrium that can be effectively computed by only looking at finite strategies and that accommodates Byzantine agents. Accordingly, in Sect. VII we give a definition of mechanism Nash equilibrium accounting for the presence of up to f Byzantine players (along the lines of [21]) and for agent *tolerance* to small ($\varepsilon > 0$) differences in rewards (along the classical lines of, e.g. [23], [26]). This leads us to the definition of ε-f-Nash equilibrium in Def. 4.

Sect. VIII gives our main theorem on which correctness of our verification algorithm rests. Theor. 2 shows that ε-f-Nash equilibria for finite state BAR systems can be automatically verified within any desired precision $\delta > 0$ by just looking at *long enough* finite sequences of actions.

Sect. IX presents our symbolic model checking algorithm for Nash equilibria. Our algorithm inputs are: a finite state mechanism $\mathcal{M}$, a proposed protocol for $\mathcal{M}$, the tolerance $\varepsilon > 0$ for agents to differences in rewards, the maximum number f of allowed Byzantine agents, our desired precision $\delta > 0$. Our algorithm returns *PASS* if the proposed protocol is indeed a $(\varepsilon + \delta)$-f-Nash equilibrium for $\mathcal{M}$, *FAIL* otherwise.

We implemented (Sect. X) our algorithm on top of NuSMV [46] using ADDs (*Arithmetic Decision Diagrams*) [18] to manipulate real valued rewards.

Finally in Sect. XI we present experimental results showing effectiveness of our approach on moderate size mechanisms. For example, within 22 hours using 5 GB of RAM we can verify mechanisms whose global present state representation requires 32 bits. The corresponding normal form games for such mechanisms would have more than 10^{22} entries.

B. Related works

Design of mechanisms for rational agents has been widely studied (e.g. [49], [45], [13]). Design methods for BAR protocols have been investigated in [1], [36], [15], [21]. We differ from such works since our focus here is on automatic verification of Nash equilibria for finite state BAR systems rather than on design principles for them.

Algorithms to search for pure, mixed (exact or approximate) Nash equilibria in games have been widely studied (e.g. see [24], [19]). We differ from such works in two ways. First, all such line of research addresses explicitly presented games (*normal form games*) whereas we are studying implicitly presented games (namely, mechanisms defined using a programming language). Thus, (because of *state explosion*) the explicit representation of a mechanism has size exponential in the size of our input. This is much the same as the relationship between reachability algorithms for directed graphs and reachability algorithms for finite state concurrent programs. As a result the algorithms and tools (e.g. [27]) for explicit games cannot be used in our context. Second, we are addressing a verification problem, thus the candidate equilibrium is an input for us whereas it is an output for the above mentioned works.

The relationship between model checking and game theory has been widely studied in many settings.

Game theoretic approaches to model checking for the verification of concurrent systems have been investigated, for example, in [32], [35], [30], [39], [51]. An example of game based model checker capable of CTL, modal μ-calculus and specification patterns is [29].

Model checking techniques have also been applied to the verification of knowledge and belief logics in game theoretic settings. Examples are in [7], [8], [12]. An example of a model checker for the logic of knowledge is MCK [28].

Applications of model checking techniques to game theory have also been investigated. For example, model checking techniques have been widely applied to the verification of games stemming from the modeling of multi-agent systems. See for example [31], [33], [38], [20]. An example of model checker for multi-agent programs is CASP [9]. An example of model checking based analysis of probabilistic games is in [5].

Note that the above papers focus on verification of temporal-like (e.g. temporal, belief, knowledge) properties of concurrent systems or of games whereas here we focus on checking Nash equilibria (of BAR protocols).

Synthesis of winning strategies for the verification game leads to automatic synthesis of correct-by-construction systems (typically controllers). This has been widely investigated in many settings. Examples are in [22], [47], [4], [2], [50], [52], [53], [3]. Note that the above papers focus on automatic synthesis (of systems or of strategies) whereas our focus here is on checking Nash equilibria (of BAR protocols).

Summing up, to the best of our knowledge, no model checking algorithm for the automatic verification of Nash equilibria of finite state mechanisms modeling BAR systems has been previously proposed.

II. Basic Notations

We denote an n-tuple of objects (of any kind) in boldface, e.g. $\boldsymbol{x}$. Unless otherwise stated we denote with x_i the i-th element of the n-tuple $\boldsymbol{x}$, $\boldsymbol{x_{-i}}$ the $(n-1)$-tuple $\langle x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \rangle$, and with $\langle \boldsymbol{x_{-i}}, x \rangle$ the n-tuple $\langle x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n \rangle$.

We denote with $\mathbb{B}$ the set $\{0, 1\}$ of boolean values (0 for *false* and 1 for *true*). We denote with $[n]$ the set $\{1, \ldots, n\}$. The set of subsets of X with cardinality at most k will be denoted by $\mathcal{P}_k(X)$.

III. Finite State Mechanisms

In this section we give the definition of *Finite State Mechanism* by suitably extending the usual definition of the synchronous parallel of finite state transition systems. This guarantees that all mechanisms consisting of finite state protocols can be modeled in our framework.

Definition 1 (Mechanism Skeleton): An n player (agent) *mechanism skeleton* $\mathcal{U}$ is a tuple $\langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ whose elements are defined as follows.

$\boldsymbol{S} = \langle S_1, \ldots, S_n \rangle$ is an n-tuple of nonempty sets (of *local states*). The *state space* of $\mathcal{U}$ is the set (of *global states*) $S = \prod_{i=1}^{n} S_i$.

$\boldsymbol{I} = \langle I_1, \ldots, I_n \rangle$ is an n-tuple of nonempty sets (of *local initial states*). The set of *global initial states* is $I = \prod_{i=1}^{n} I_i$.

$\boldsymbol{A} = \langle A_1, \ldots, A_n \rangle$ is an n-tuple of nonempty sets (of *local actions*). The set of *global actions* (i.e. n-tuples of local actions) is $A = \prod_{i=1}^{n} A_i$. The set of i*-opponents actions* is $A_{-i} = \prod_{j=1, j \neq i}^{n} A_i$.

$\boldsymbol{B} = \langle B_1, \ldots, B_n \rangle$ is an n-tuple of functions s.t., for each $i \in [n]$, $B_i : S \times A_i \times S_i \to \mathbb{B}$. Function B_i models the *transition relation* of agent i, i.e. $B_i(\boldsymbol{s}, a, s')$ is true iff agent i can move from (global) state $\boldsymbol{s}$ to (local) state s' via action a. We require B_i to be *serial* (i.e. $\forall \boldsymbol{s} \in S\ \exists a \in A_i\ \exists s' \in S_i$ s.t. $B_i(\boldsymbol{s}, a, s')$ holds) and *deterministic* (i.e. $B_i(\boldsymbol{s}, a, s') \wedge B_i(\boldsymbol{s}, a, s'')$ implies $s' = s''$). We write $B_i(\boldsymbol{s}, a)$ for $\exists s'\ B_i(\boldsymbol{s}, a, s')$. That is, $B_i(\boldsymbol{s}, a)$ holds iff action a is allowed in state $\boldsymbol{s}$ for agent i. For each agent $i \in [n]$, function B_i models the *underlying behavior* of agent i. That is, the set of all possible choices of *rational* player i. As a result, B_i defines the transition relation for the *Byzantine* behavior for agent i.

$\boldsymbol{h} = \langle h_1, \ldots, h_n \rangle$ is an n-tuple of functions s.t., for each player $i \in [n]$, $h_i : S \times A \to \mathbb{R}$. Function h_i models the *payoff (reward) function* of player i. Note that $\boldsymbol{h}$ may be seen as a function $\boldsymbol{h} : S \times A \to \mathbb{R}^n$ s.t. $\boldsymbol{h}(\boldsymbol{s}, \boldsymbol{a}) = (h_1(\boldsymbol{s}, \boldsymbol{a}), \ldots, h_n(\boldsymbol{s}, \boldsymbol{a}))$ for all global states $\boldsymbol{s} \in S$ and global actions $\boldsymbol{a} \in A$.

$\boldsymbol{\beta} = \langle \beta_1, \ldots, \beta_n \rangle$ is an n-tuple of *discounts*, that is of real values such that for each $i \in [n]$, $\beta_i \in (0, 1)$.

Definition 2 (Mechanism): An n player *mechanism* $\mathcal{M}$ is a pair $(\mathcal{U}, \boldsymbol{T})$ where: $\mathcal{U} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ is a mechanism skeleton and $\boldsymbol{T} = \langle T_1, \ldots, T_n \rangle$ is an n-tuple of functions s.t., for each $i \in [n]$, $T_i : S \times A_i \to \mathbb{B}$. We require T_i to satisfy the following properties: 1) $T_i(\boldsymbol{s}, a)$ implies $B_i(\boldsymbol{s}, a)$; 2) (*nonblocking*) for each state $\boldsymbol{s} \in \boldsymbol{S}$ there exists an action $a \in A_i$ s.t. $T_i(\boldsymbol{s}, a)$ holds. Function T_i models the *proposed protocol* for agent i, that is its *obedient* (or *altruistic*, following [1], [36]) behavior. More specifically, if agent i is *altruistic* then its transition relation is $B_i(\boldsymbol{s}, a, s') \wedge T_i(\boldsymbol{s}, a)$.

Often we denote an n player mechanisms $\mathcal{M}$ with the tuple $\langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$. Furthermore we may also call mechanism a mechanism skeleton. The context will always make clear the intended meaning.

Remark 1 (Finite State Agents): In order to develop our model checking algorithm we model each agent as a *Finite State Machine* (FSM). This limits agent knowledge about the past. In fact, the system *state* represents the system past history. Since our systems are nonterminating ones, histories (and thus agent knowledge) are in general unbounded. As for verification of security protocols (e.g. see [43], [44]) it is the modeler responsibility to develop a suitable finite state approximation of knowledge.

Definition 3: Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$, be an n player mechanism and $Z \subseteq [n]$. Let $BT : \mathcal{P}([n]) \times \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S} \to \mathbb{B}$ be such that $BT(Z, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}) = \bigwedge_{i=1}^{n} BT_i(Z, \boldsymbol{s}, a_i, s'_i)$, where

$$BT_i(Z, \boldsymbol{s}, a_i, s'_i) = \begin{cases} B_i(\boldsymbol{s}, a_i, s'_i) & \text{if } i \in Z \\ B_i(\boldsymbol{s}, a_i, s'_i) \wedge T_i(\boldsymbol{s}, a_i) & \text{otherwise.} \end{cases}$$

BT models the transition relation of mechanism $\mathcal{M}$, when the set of Byzantine players is Z and all agents not in Z are altruistic.

IV. An Example of Mechanism

In order to clarify our definitions we give an example of a simple mechanism. Consider the situation in which a set of agents cooperate to accomplish a certain job. The job, in turn, consists of n tasks. Each task is assigned to at least one agent which may carry out the assigned task or may deviate by not doing any work. Carrying out the assigned task entails a cost (negative reward) for the agent. On the other hand, if all tasks forming the job are completed (and thus the job itself is completed) all agents that have worked to a task get a reward greater than the cost incurred to carry out the assigned task. If the job is not completed no agent gets anything. The mechanism skeleton (Def. 1) is defined as follows.

All agents have the same discount, say $\beta = 0.5$, and the same underlying (Byzantine) behavior, defined by the automaton B_i in Fig. 1.

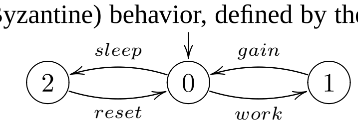

Fig. 1. Underlying behavior B_i for agent i.

From Fig. 1 we have: $S_i = \{0, 1, 2\}$, $I_i = \{0\}$, $A_i = \{\text{reset}, \text{sleep}, \text{work}, \text{gain}\}$. Let $\boldsymbol{S} = \langle S_1, \ldots, S_n \rangle$, $\boldsymbol{s} = \langle s_1, \ldots, s_n \rangle \in S$ and $\boldsymbol{a} = \langle a_1, \ldots, a_n \rangle \in A$.

The mechanism skeleton is $\mathcal{U} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ where the payoff function h_i for agent i is defined as follows:

$$h_i(\boldsymbol{s}, \langle \boldsymbol{a}_{-i}, \text{work} \rangle) = -1$$

$$h_i(\boldsymbol{s}, \langle \boldsymbol{a}_{-i}, \text{sleep} \rangle) = h_i(\boldsymbol{s}, \langle \boldsymbol{a}_{-i}, \text{reset} \rangle) = 0$$

$$h_i(\boldsymbol{s}, \langle \boldsymbol{a}_{-i}, \text{gain} \rangle) = \begin{cases} 4 & \text{if } (s_i = 1) \text{ for all } i \in [n] \\ 0 & \text{otherwise} \end{cases}$$

The mechanism (Def. 2) is $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$. where the *proposed protocol* T_i for agent i requires agent i to cooperate, that is to carry out the assigned task. Formally: $T_i(\boldsymbol{s}, a_i) = ((s_i = 0) \wedge (a_i = \text{work})) \vee ((s_i = 1) \wedge (a_i = \text{gain})) \vee ((s_i = 2) \wedge (a_i = \text{reset}))$.

V. Paths in Mechanisms

Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ be an n player mechanism and let $Z \subseteq [n]$ be a set of (Byzantine) agents.

A *path* in ($\mathcal{M}$, Z) (or simply a *path* when ($\mathcal{M}$, Z) is understood from the context) is a (finite or infinite) sequence $\pi = \boldsymbol{s}(0)\boldsymbol{a}(0)\boldsymbol{s}(1)\ldots\boldsymbol{s}(t)\boldsymbol{a}(t)\boldsymbol{s}(t+1)\ldots$ where, for each t, $\boldsymbol{s}(t)$ is a global state, $\boldsymbol{a}(t)$ is a global action and $BT(Z, \boldsymbol{s}(t), \boldsymbol{a}(t), \boldsymbol{s}(t+1))$ holds.

The *length* of a path is the number of global actions in a path. We denote with $|\pi|$ the *length* of path π. If π is infinite we write $|\pi| = \infty$. Note that if $\pi = \boldsymbol{s}(0)$ then $|\pi| = 0$. Thus a path of length 0 is not empty.

In order to extract the t-th global state and the t-th global action from a given path π, we define $\pi^{(s)}(t) = \boldsymbol{s}(t)$ and $\pi^{(a)}(t) = \boldsymbol{a}(t)$. To extract local actions, we denote with $\pi_i^{(a)}(t)$ the action $a_i(t)$ at stage t of agent i and with $\pi_{-i}^{(a)}(t)$ the actions $\boldsymbol{a_{-i}}(t)$ at stage t of all agents but i.

For each agent $i \in [n]$, the *value* of a path π is $v_i(\pi) = \sum_{t=0}^{|\pi|-1} \beta_i^t h_i\left(\pi^{(s)}(t), \pi^{(a)}(t)\right)$. Note that for any path π and agent $i \in [n]$ the *path value* $v_i(\pi)$ is well defined also when $|\pi| = \infty$ since the series $\sum_{t=0}^{\infty} \beta_i^t h_i\left(\pi^{(s)}(t), \pi^{(a)}(t)\right)$ converges for all $\beta_i \in (0, 1)$. The *path value vector* is defined as: $\boldsymbol{v}(\pi) = \langle v_1(\pi), \ldots, v_n(\pi) \rangle$.

A *path* π in ($\mathcal{M}$, Z) is said to be i-*altruistic* if for all $t < |\pi|$, $T_i(\pi^{(s)}(t), \pi_i^{(a)}(t))$ holds.

Given a path π and a nonnegative integer $k \leq |\pi|$ we denote with $\pi|_k$ the *prefix* of π of length k, i.e. the finite path $\pi|_k = \boldsymbol{s}(0)\boldsymbol{a}(0)\boldsymbol{s}(1)\ldots\boldsymbol{s}(k)$ and with $\pi|^k$ the *tail* of π, i.e. the path $\pi|^k = \boldsymbol{s}(k)\boldsymbol{a}(k)\boldsymbol{s}(k+1)\ldots\boldsymbol{s}(t)\boldsymbol{a}(t)\boldsymbol{s}(t+1)\ldots$.

We denote with $\mathrm{Path}_k(\boldsymbol{s}, Z)$ the set of all paths of length k starting at $\boldsymbol{s}$. Formally, $\mathrm{Path}_k(\boldsymbol{s}, Z) = \{\pi \mid \pi$ is a path in ($\mathcal{M}$, Z) and $|\pi| = k$ and $\pi^{(s)}(0) = \boldsymbol{s}\}$.

We denote with $\mathrm{Path}_k(\boldsymbol{s}, f)$ the set of all paths of length k feasible with respect to all sets of Byzantine agents of cardinality at most f. Formally, $\mathrm{Path}_k(\boldsymbol{s}, f) = \bigcup_{|Z| \leq f} \mathrm{Path}_k(\boldsymbol{s}, Z)$. We write $\mathrm{Path}_k(\boldsymbol{s})$ for $\mathrm{Path}_k(\boldsymbol{s}, n)$.

Unless otherwise stated in the following, we omit the subscript or superscript horizon when it is ∞. For example, we write $\mathrm{Path}(\boldsymbol{s}, Z)$ for $\mathrm{Path}_\infty(\boldsymbol{s}, Z)$.

Note that if $i \notin Z$, all paths in $\mathrm{Path}_k(\boldsymbol{s}, Z)$ are i-altruistic, that is, agent i behaves accordingly to the proposed protocol.

VI. Strategies

Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ be an n player mechanism and let $Z \subseteq [n]$ be a set of (Byzantine) agents.

As usual in a game theoretic setting, we need to distinguish player actions (i.e. local actions) from those of its opponents. This leads to the notion of strategy.

A *strategy* σ is a (finite or infinite) sequence of local actions for a given player. The *length* $|\sigma|$ of σ is the number of actions in σ (thus if $|\sigma| = 0$, the strategy is empty).

A strategy σ for player i *agrees* with a path π (notation $\pi \simeq^i \sigma$) if $|\sigma| = |\pi|$ and for all $t < |\sigma|$, $\sigma(t) = \pi_i^{(a)}(t)$.

Given a path π, the strategy (of length $|\pi|$) for player i associated to π is $\sigma(\pi, i) = \pi_i^{(a)}(0)\pi_i^{(a)}(1)\ldots\pi_i^{(a)}(t)\ldots$.

The set of Z-*feasible strategies* of length k for player i in state $\boldsymbol{s}$ is: $\mathrm{Strat}_k(\boldsymbol{s}, Z, i) = \{\sigma(\pi, i) \mid \pi \in \mathrm{Path}_k(\boldsymbol{s}, Z)\}$.

The set of f-*feasible strategies* of length k for player i in state $\boldsymbol{s}$ is: $\mathrm{Strat}_k(\boldsymbol{s}, f, i) = \{\sigma(\pi, i) \mid \pi \in \mathrm{Path}_k(\boldsymbol{s}, f)\}$.

As for paths, a strategy $\sigma \in \mathrm{Strat}_k(\boldsymbol{s}, Z, i)$ is said to be i-*altruistic* if $i \notin Z$. We use the notations $\sigma|_k$ and $\sigma|^k$ to denote, respectively, the k-prefix and the tail after k steps of a strategy.

The set of paths that agree with a set of strategies Σ for an agent i is defined as follows. $\mathrm{Path}(\boldsymbol{s}, Z, i, \Sigma) = \{\pi \in \mathrm{Path}_k(\boldsymbol{s}, Z) \mid \exists \sigma \in \Sigma.\ k = |\sigma| \wedge \pi \simeq^i \sigma\}$. When Σ is the singleton $\{\sigma\}$, we simply write $\mathrm{Path}(\boldsymbol{s}, Z, i, \sigma)$.

The *guaranteed outcome* (or the *value*) of a strategy σ in state $\boldsymbol{s}$ for player i is the minimum value of paths that agree with σ. Formally: $v_i(Z, \boldsymbol{s}, \sigma) = \min\{v_i(\pi) \mid \pi \in \mathrm{Path}(\boldsymbol{s}, Z, i, \sigma)\}$

The *value* of a state $\boldsymbol{s}$ at horizon k for player i is the guaranteed outcome of the best strategy of length k starting at state $\boldsymbol{s}$. Formally: $v_i^k(Z, \boldsymbol{s}) = \max\{v_i(Z, \boldsymbol{s}, \sigma) \mid \sigma \in \mathrm{Strat}_k(\boldsymbol{s}, Z, i)\}$.

The *worst case value* of a state $\boldsymbol{s}$ at horizon k for player i is the outcome of the worst strategy of length k starting at state $\boldsymbol{s}$. Formally, $u_i^k(Z, \boldsymbol{s}) = \min\{v_i(Z, \boldsymbol{s}, \sigma) | \sigma \in \mathrm{Strat}_k(\boldsymbol{s}, Z, i)\}$.

As usual, we will write $v_i(Z, \boldsymbol{s})$ for $v_i^\infty(Z, \boldsymbol{s})$, and $u_i(Z, \boldsymbol{s})$ for $u_i^\infty(Z, \boldsymbol{s})$.

The finite horizon value of a state can be effectively computed by using a dynamic programming approach (Prop. 1). This is one of the main ingredients of our verification algorithm (Sect. IX). We omit proofs because of lack of space.

Proposition 1: Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ be an n player mechanism, $i \in [n]$, $Z \subseteq [n]$ and $\boldsymbol{s} \in S$. The state values at horizon k for player i can be computed as follows:

- $v_i^0(Z, \boldsymbol{s}) = u_i^0(Z, \boldsymbol{s}) = 0$;
- $v_i^{k+1}(Z, \boldsymbol{s}) = \max_{a_i \in A_i} \min_{\boldsymbol{a}_{-i} \in A_{-i}} \{h_i(\boldsymbol{s}, \langle \boldsymbol{a}_{-i}, a_i \rangle) + \beta_i\, v_i^k(Z, \boldsymbol{s'}) \mid BT(Z, \boldsymbol{s}, \langle \boldsymbol{a}_{-i}, a_i \rangle, \boldsymbol{s'})\}$
- $u_i^{k+1}(Z, \boldsymbol{s}) = \min_{a_i \in A_i} \min_{\boldsymbol{a}_{-i} \in A_{-i}} \{h_i(\boldsymbol{s}, \langle \boldsymbol{a}_{-i}, a_i \rangle) + \beta_i\, u_i^k(Z, \boldsymbol{s'}) \mid BT(Z, \boldsymbol{s}, \langle \boldsymbol{a}_{-i}, a_i \rangle, \boldsymbol{s'})\}$

VII. Nash Equilibria in Mechanisms

Our notion of *Nash equilibrium* for a mechanism combines those in [23], [21]. Intuitively, a mechanism $\mathcal{M}$ is ε-f-Nash, if as long as the number of Byzantine agents is no more than f (e.g. see [21]), no rational agent has an interest greater than ε (e.g. see [23], [26]) in deviating from the proposed protocol in $\mathcal{M}$.

Definition 4 (ε-f-Nash): Let $\mathcal{M} = \langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ be an n player mechanism, $f \in \{0, \ldots, n\}$ and $\epsilon > 0$. $\mathcal{M}$ is ε-f-Nash for player $i \in [n]$ if $\forall Z \in \mathcal{P}_f([n] \setminus \{i\})$, $\forall \boldsymbol{s} \in I$, $u_i(Z, \boldsymbol{s}) + \varepsilon \geq v_i(Z \cup \{i\}, \boldsymbol{s})$. $\mathcal{M}$ is ε-f-Nash if it is ε-f-Nash for each player $i \in [n]$.

Note that ε-0-Nash is the ε-Nash equilibrium defined in [23], [26]. Furthermore, stretching Def. 4 by setting $\varepsilon = 0$, we see that 0-f-Nash is the f-Nash equilibrium defined in

[21] whereas 0-0-Nash is the classical Nash equilibrium (e.g., see [26]).

Observe that, for each agent i, we compare agent i best reward when it considers deviating from the protocol ($v_i(Z \cup \{i\}, \boldsymbol{s})$), with agent i worst reward when it obeys the protocol ($u_i(Z, \boldsymbol{s})$). The reason for tolerating a small (ε) tolerance on rewards when deviating from the proposed protocol in Def. 4 is that our aim is to verify Nash equilibria by looking only at finite strategies. It is well known (e.g. see Sect. 4.8 of [26]) that ε-0-Nash equilibria have been introduced to get within a finite horizon equilibria that are only available with an infinite horizon. This means that a finite horizon may not suffice to check that a mechanism is 0-0-Nash. The following example aims at clarifying the above well known game-theoretical issue framing it in our context.

Example 1: Let $\mathcal{M}$ be a one agent (named 1) mechanism defined as follows. The underlying behavior B_1 for agent 1 is shown in Fig. 2 where on the automaton edges we show action names as well as payoff values since in this simple case the payoff function depends only on local states and local actions of the agent. The discount factor for agent 1 is $\beta_1 = \frac{1}{2}$. Let the *proposed protocol* $\boldsymbol{T}$ of $\mathcal{M}$ be defined as follows: $T_1(\boldsymbol{s}, \boldsymbol{a})$ = $((\boldsymbol{s} = 0) \wedge (\boldsymbol{a} = \text{a})) \vee ((\boldsymbol{s} = 1) \wedge (\boldsymbol{a} = \text{d})) \vee ((\boldsymbol{s} = 2) \wedge (\boldsymbol{a} = \text{e})) \vee (\boldsymbol{s} \geq 3)$. We focus on the case $f = 0$, that is there are no Byzantine agents and hence $Z = \varnothing$. For all $k > 0$ we have: $u_1^k(\varnothing, 0) = -1 + \frac{3}{2}\sum_{i=0}^{k-2}(-\frac{1}{2})^i$=$(-1)^k \frac{1}{2^{k-1}}$ (the protocol $\boldsymbol{T}$ prescribes to follow the strategy $a(de)^\omega$) and $v_1^k(\{1\}, 0)$ = $\frac{1}{2^{k-1}}$ (v_1 will use strategy $\sigma_1 = a(de)^\omega$ when k is even and $\sigma_2 = c(gh)^\omega$ when k is odd). Therefore, if k is odd $u_1^k(\varnothing, 0) < v_1^k(\{1\}, 0)$, and if k is even $u_1^k(\varnothing, 0) = v_1^k(\{1\}, 0)$, Thus there is no $\bar{k} > 0$ s.t. for all $k \geq \bar{k}$, $u_1^k(\varnothing, 0) \geq v_1^k(\{1\}, 0)$.

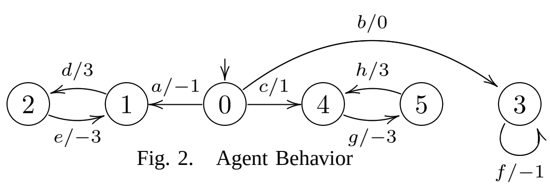

Fig. 2. Agent Behavior

Finding ε-0-Nash equilibria even for finite horizon games is not trivial (e.g. see [19]). As for infinite games, we note that game-theory results focus on showing that an ε *small enough* can be found so that an infinite horizon 0-0-Nash equilibrium becomes a finite horizon ε-0-Nash one (e.g. see [26]). Our concern here is different. We are given ε (fixed) and want to verify if the given mechanism is ε-0-Nash (actually, ε-f-Nash).

From Example 1 one may conjecture that ε-f-Nash ($\varepsilon > 0$) equilibria can be verified by just looking at *long enough* finite horizons. Unfortunately this is not always the case as shown by the following example.

Example 2: Consider again the mechanism $\mathcal{M}$ in Example 1. Let the *proposed protocol* $\boldsymbol{T}$ of $\mathcal{M}$ be defined as follows: $T_1(\boldsymbol{s}, \boldsymbol{a})$ = $((\boldsymbol{s} = 0) \wedge (\boldsymbol{a} = \text{b})) \vee (\boldsymbol{s} \geq 1)$. Also in this case we focus on the case in which there are no Byzantine agents, that is $f = 0$ and $Z = \varnothing$. For all $k > 0$ we have: $u_1^k(\varnothing, 0)$ = $\sum_{i=1}^{k} -\frac{1}{2^i}$ = $-1 + \frac{1}{2^k}$ and $v_1^k(\{1\}, 0)$ = $\frac{1}{2^{k-1}}$. For all k we have: $\Delta(k)$ = $v_1^k(\{1\}, 0) - u_1^k(\varnothing, 0)$ = $(1 + \frac{1}{2^k})$. Now, $\mathcal{M}$ is clearly 1-0-Nash however there is no $\bar{k} > 0$ s.t. for all $k \geq \bar{k}$, $\Delta(k) \leq 1$. That is, there is no finite horizon that allows us to conclude that $\mathcal{M}$ is 1-0-Nash. Note, however, that for all $\delta > 0$ there exists a $\bar{k} > 0$ s.t. for all $k \geq \bar{k}$, $\Delta(k) \leq 1 + \delta$. Thus, for all $\delta > 0$, by just considering a suitable finite horizon $\bar{k} > 0$, we can verify that $\mathcal{M}$ is $(1 + \delta)$-0-Nash.

VIII. Verifying ε-f-Nash Equilibria

In this Section we give our main theorem (Theor. 2) on which correctness of our verification algorithm (Sect. IX) rests.

Example 2 shows that, in general, using finite horizon approximations, we cannot verify ε-f-Nash equilibria. However the very same example suggests that we may get arbitrarily close to this result. This is indeed our main theorem.

Theorem 2 (Main Theorem): Let $\mathcal{M}$ = $\langle \boldsymbol{S}, \boldsymbol{I}, \boldsymbol{A}, \boldsymbol{T}, \boldsymbol{B}, \boldsymbol{h}, \boldsymbol{\beta} \rangle$ be an n player mechanism, $f \in \{0, \ldots, n\}$, $\varepsilon > 0$ and $\delta > 0$. Furthermore, for each agent $i \in [n]$ let:

1) M_i = $\max\{|h_i(\boldsymbol{s}, \boldsymbol{a})| \mid \boldsymbol{s} \in S$ and $\boldsymbol{a} \in A\}$.
2) $E_i(k)$ = $5\, \beta_i^k \frac{M_i}{1-\beta_i}$.
3) $\Delta_i(k)$ = $\max\{v_i^k(Z \cup \{i\}, \boldsymbol{s}) - u_i^k(Z, \boldsymbol{s}) \mid \boldsymbol{s} \in I, Z \in \mathcal{P}_f([n] \setminus \{i\})\}$
4) $\varepsilon_1(i, k)$ = $\Delta_i(k) - 2E_i(k)$
5) $\varepsilon_2(i, k)$ = $\Delta_i(k) + 2E_i(k)$

For each agent i, let k_i be s.t. $4\, E_i(k_i) < \delta$. Then we have:

1) If for each $i \in [n]$, $\varepsilon \geq \varepsilon_2(i, k_i) > 0$ then $\mathcal{M}$ is ε-f-Nash.
2) If there exists $i \in [n]$ s.t. $0 < \varepsilon \leq \varepsilon_1(i, k_i)$ then $\mathcal{M}$ is not ε-f-Nash. Of course in such a case a fortiori $\mathcal{M}$ is not 0-f-Nash.
3) If for each $i \in [n]$, $\varepsilon_1(i, k_i) < \varepsilon$ and there exists $j \in [n]$ s.t. $\varepsilon < \varepsilon_2(j, k_j)$ then $\mathcal{M}$ is $(\varepsilon + \delta)$-f-Nash.

Proof: Because of lack of space we omit the proof. See [42] for the complete proof. ■

IX. ε-f-Nash Verification Algorithm

Resting on Prop. 1 and on Theor. 2, Algorithm 1 verifies that a given n agent mechanism $\mathcal{M}$ is ε-f-Nash.

In Algorithm 1, $\boldsymbol{s}$ and $\boldsymbol{a}$ are vectors (of boolean variables) ranging respectively on (the boolean encoding of) states ($\boldsymbol{S}$) and actions ($\boldsymbol{A}$).

The set Z of Byzantine agents in Algorithm 1 can also be represented with a vector of n boolean variables $\mathbf{b}$ = $\langle b_1, \ldots b_n \rangle$ such that for each agent $i \in [n]$, agent i is Byzantine iff $b_i = 1$. Accordingly, the constraint $Z \in \mathcal{P}_f([n] \setminus \{i\})$ in Def. 4 becomes a constraint on $\mathbf{b}$, namely: $(\sum_{i=1}^{i=n} b_i \leq f \wedge b_i = 0)$. Along the same lines, the set $Z \cup \{i\}$ (used in Algorithm 1) can be represented with the boolean vector $\mathbf{b}[b_i := 1]$ obtained from $\mathbf{b}$ by replacing variable b_i in $\mathbf{b}$ with the boolean constant 1. For readability in Algorithm 1 we use Z rather than its boolean representation $\mathbf{b}$.

First of all, in line 3 of Algorithm 1, we compute the horizon k needed for agent i to achieve the required accuracy δ.

Lines 4–11 use Prop. 1 to compute state values at horizon k of state $\boldsymbol{s}$ with the set of Byzantine players Z when player i obeys the protocol ($u_i^k(Z, \boldsymbol{s})$) as well as when player i behaves arbitrarily within the underlying behavior ($v_i^k(Z \cup \{i\}, \boldsymbol{s})$).

Line 12 computes the max difference between initial state values for a rational player and those for a player following the proposed protocol, by also maximizing over all $Z \in \mathcal{P}_f([n] \setminus \{i\})$. (hypothesis 3 of Theor. 2, see also Def. 4). Line 13 computes the values in hypotheses 4 and 5 of Theor. 2.

Line 14 returns *FAIL* as soon as the hypothesis of thesis 2 of Theor. 2 is satisfied. In such a case from Theor. 2 we know that the given mechanism is not ε-f-Nash and thus it is not f-Nash.

Line 15 (16) returns, *PASS with* ε (*PASS with* $\varepsilon + \delta$) when the hypothesis of thesis 1 (3) of Theor. 2 is satisfied. In such a case from Theor. 2 we know that the given mechanism is ε-f-Nash (($\varepsilon + \delta$)-f-Nash).

Algorithm 1 Checking if a mechanism is ε-f-Nash

1: *CheckNash*(mechanism $\mathcal{M}$, int f, double ε, δ)
2: **for all** $i \in [n]$ **do**
3: Let k such that $4\,E_i(k) < \delta$
4: Let $\boldsymbol{s} \in \boldsymbol{S}$ and $Z \in \mathcal{P}_f([n] \setminus \{i\})$
5: $v_i^0(Z, \boldsymbol{s}) \leftarrow 0$; $u_i^0(Z, \boldsymbol{s}) \leftarrow 0$;
6: **for** $t = 1$ **to** k **do**
7: $v_i^t(Z \cup \{i\}, \boldsymbol{s}) \leftarrow \max\limits_{a_i \in A_i} \min\limits_{\boldsymbol{a}_{-i} \in A_{-i}} [h_i(\boldsymbol{s}, \langle a_i, \boldsymbol{a}_{-i}\rangle) +$
8: $+\beta_i v_i^{t-1}(\boldsymbol{s'}, Z \cup \{i\})]$,
9: $BT(Z \cup \{i\}, \boldsymbol{s}, \langle a_i, \boldsymbol{a}_{-i}\rangle, \boldsymbol{s'})$
10: $u_i^t(Z, \boldsymbol{s}) \leftarrow \min\limits_{a_i \in A_i} \min\limits_{\boldsymbol{a}_{-i} \in A_{-i}} [h_i(\boldsymbol{s}, \langle a_i, \boldsymbol{a}_{-i}\rangle) +$
11: $+\beta_i u_i^{t-1}(Z, \boldsymbol{s'})], BT(Z, \boldsymbol{s}, \langle a_i, \boldsymbol{a}_{-i}\rangle, \boldsymbol{s'})$
12: $\Delta_i \leftarrow \max\{v_i^k(Z \cup \{i\}, \boldsymbol{s}) - u_i^k(Z, \boldsymbol{s}) \mid \boldsymbol{s} \in I, Z \in \mathcal{P}_f([n] \setminus \{i\})\}$
13: $\varepsilon_1(i) \leftarrow \Delta_i - 2E_i(k)$; $\varepsilon_2(i) \leftarrow \Delta_i + 2E_i(k)$
14: **if** ($\varepsilon < \varepsilon_1(i)$) **return** (FAIL)
15: **if** ($\forall i \in [n](\varepsilon_2(i) < \varepsilon)$) **return** (PASS with ε)
16: **else return** (PASS with $(\varepsilon + \delta)$)

X. Implementation

We implemented Algorithm 1 within the NuSMV [46] model checker. Here we briefly describe the main ideas bridging the gap between Algorithm 1 and its NuSMV implementation.

First of all, we extended the SMV language so as to be able to define mechanisms. We should keep in mind that, once we have verified that the proposed protocol is a Nash equilibrium for the given mechanism, then we will undertake a standard CTL verification in order to check that the proposed protocol satisfies the desired safety and liveness properties. Accordingly, we confined most of our extensions to the SMV language inside SMV comments so that mechanism models can also be used for standard CTL verification again with NuSMV.

As a second step we implemented Algorithm 1 using *Ordered Binary Decision Diagrams* (OBDDs) [10] resting on the CUDD [18] OBDD package (which is also the one used in NuSMV). Note that all functions in Algorithm 1 depend only on boolean variables, namely those representing states, actions and sets of Byzantine agents.

From Algorithm 1 we see that we only have two kinds of functions: *b2b* functions, taking booleans and returning a boolean value, and *b2r* functions, taking booleans and returning a real value. As usual *b2b* functions can be effectively represented using OBDDs. As for *b2r* functions we used the *Arithmetic Decision Diagrams* (ADDs) available in the CUDD package. ADDs are designed to represent and efficiently manipulate *b2r* functions returning reals represented as `C` 64 bit double. All arithmetical operations on ADDs used in Algorithm 1 are available in the CUDD package. The only ones that we had to implement ourselves were the $\max$ and $\min$ functions on ADDs. We developed them with a suitable traversal of the ADD to be minimized (or maximized). See [42] for a full description of our symbolic implementation of Algorithm 1.

XI. Experimental Results

In order to assess effectiveness of our Nash verifier we present experimental results on its usage on an n player mechanism $\mathcal{M}$ generalizing the mechanism presented in Section IV. This is a meaningful and scalable case study that well serves our purposes.

A. Mechanism Description

We are given a set $\mathcal{J} = \{0, \ldots m-1\}$ of m *jobs* and a set $\mathcal{T} = \{0, \ldots q-1\}$ of q *tasks*. Function $\eta : \mathcal{J} \to \mathcal{P}(\mathcal{T})$ defines for each job j the set of tasks $\eta(j)$ needed to complete j.

Each agent $i \in [n]$ is supposed to work (*proposed protocol*) on a given sequence of (not necessarily distinct) tasks $\mathcal{T}_i = \langle \tau(i, 0), \ldots \tau(i, \alpha(i) - 1)\rangle$ starting from $\tau(i, 0)$ and returning to $\tau(i, 0)$ after task $\tau(i, \alpha(i) - 1)$ has been completed. An agent may *deviate* from the proposed protocol by delaying execution of a task or by not executing the task at all. This models many typical scenarios in cooperative services.

An agent incurs a *cost* by working towards the completion of its currently assigned task. Once an agent has completed a task it waits for its reward (if any) before it considers working to the next task in its list. As soon as an agent receives its reward it considers working to the next task in its list.

A job is completed if for each task it needs, there exists at least one agent that has completed that task. In such a case, each of such agents receive a *reward*. Note that even if two (or more) agents have completed the same task, all of them get a reward.

Agent i can be modeled as follows. Its set of states is $X_i = Y_i \times Z_i$, where: $Y_i = \{0, \ldots (\alpha(i) - 1)\}$ and $Z_i = \{0, 1, 2\}$. State variable $\mathbf{y}_i$ ranges on Y_i, state variable $\mathbf{z}_i$ ranges on Z_i.

State variable $\mathbf{z}_i$ models the working state of agent i. Namely, $\mathbf{z}_i = 0$ if agent i is not working; $\mathbf{z}_i = 1$ if agent i has done its assigned work (that is it has completed its currently assigned task); $\mathbf{z}_i = 2$ if agent i currently completed task has been used to complete a job. In the last case agent i gets its reward for having completed a task used by a job. State variable $\mathbf{y}_i$ keeps track of the task currently assigned to agent i. That is, $\mathbf{y}_i = p$ iff agent i is supposed to complete task $\tau(i, p)$ in its sequence of assigned tasks.

Agent i's set of initial states is I_i = $\{(0,0)\}$ whereas agent i's set of actions is A_i = $\{0,1\}$ with variable $\mathbf{a}_i$ ranging on it. Variable $\mathbf{a}_i$ models agent i's choices. Namely, $\mathbf{a}_i = 1$ if agent i will work. and $\mathbf{a}_i = 0$ otherwise.

The state space of $\mathcal{M}$ is $\boldsymbol{S}$ = $\langle Y_1,\, Z_1,\, \ldots\, Y_n,\, Z_n\rangle$. The set of initial states of $\mathcal{M}$ is $\boldsymbol{I}$ = $\langle I_1,\, \ldots\, I_n\rangle$ We denote with $\mathbf{s}$ the vector (of present state variables) $\langle \mathbf{y}_1,\, \mathbf{z}_1,\, \ldots\, \mathbf{y}_n,\, \mathbf{z}_n\rangle$ and with $\mathbf{a}$ the vector (of action variables) $\langle \mathbf{a}_1,\, \ldots\, \mathbf{a}_n\rangle$.

Let $\Phi(i)$ be the set of pairs $(j,\, p)$ such that the p-th task of agent i task sequence is needed for job j. Formally, $\Phi(i)$ = $\{(j,p) \mid (p \in \{0, \ldots \alpha(i)-1\} \wedge (j \in \mathcal{J}) \wedge (\tau(i,p) \in \eta(j))\}$.

Let $\Gamma(t)$ be the set of pairs $(w,\, r)$ such that the r-th task of agent w task sequence is t. Formally, $\Gamma(t)$ = $\{(w,r) \mid (w \in [n]) \wedge (r \in \{0, \ldots \alpha(w)-1\}) \wedge (t = \tau(w,r))\}$.

Let $\varphi_j(\mathbf{s})$ be a boolean function which is true iff job j is completed. That is, if for each task t in job j there exists an agent w such that w has completed task t. Formally, $\varphi_j(\mathbf{s})$ = $\wedge_{t \in \eta(j)} \vee_{(w,r) \in \Gamma(t)} ((\mathbf{y}_w = r) \wedge (\mathbf{z}_w = 1))$.

Let $\gamma_i(\mathbf{s})$ be a boolean function which is true iff agent i is currently assigned to a task needed for a currently completed job. Formally, $\gamma_i(\mathbf{s})$ = $(\vee_{(j,p) \in \Phi(i)}\ ((\mathbf{y}_i = p) \wedge (\varphi_j(\mathbf{s}))))$.

Finally, the underlying behavior B_i of agent i is defined as follows: $B_i(\mathbf{s},\, \mathbf{a}_i,\, \mathbf{y}_i',\, \mathbf{z}_i')$ =

$((\mathbf{z}_i = 0) \wedge (\mathbf{a}_i = 0) \wedge (\mathbf{y}_i' = \mathbf{y}_i) \wedge (\mathbf{z}_i' = \mathbf{z}_i)) \vee$
$((\mathbf{z}_i = 0) \wedge (\mathbf{a}_i = 1) \wedge (\mathbf{y}_i' = \mathbf{y}_i) \wedge (\mathbf{z}_i' = 1)) \vee$
$((\mathbf{z}_i = 1) \wedge \gamma_i(\mathbf{s}) \wedge (\mathbf{z}_i' = 2) \wedge (\mathbf{y}_i' = \mathbf{y}_i)))) \vee$
$((\mathbf{z}_i = 1) \wedge \neg\gamma_i(\mathbf{s}) \wedge (\mathbf{z}_i' = 1) \wedge (\mathbf{y}_i' = \mathbf{y}_i)))) \vee$
$((\mathbf{z}_i = 2) \wedge (\mathbf{y}_i' = (\mathbf{y}_i + 1) \bmod \alpha(i)) \wedge (\mathbf{z}_i' = 0))$.

The proposed protocol T_i for agent i is $T_i(\mathbf{s},\, \mathbf{a}_i) = (\mathbf{a}_i = 1)$, that is agent i is supposed to carry out the assigned task as soon as it can. Reward h_i for agent i is defined as follows:

$$h_i(\mathbf{s}, \mathbf{a}_i) = \begin{cases} -1 & \text{if } ((\mathbf{z}_i = 0) \wedge (\mathbf{a}_i = 1)) \\ +4 & \text{if } (\mathbf{z}_i = 2) \\ 0 & \text{otherwise} \end{cases}$$

B. Experimental Settings

In order to run our Nash verification experiments we instantiate the above class of mechanisms as follows. First of all, we take the number of agents (n) to be greater than or equal to that of tasks (q). Second, we take the number of jobs (m) to be equal to the number of tasks (q). Third, we define $\eta(j)$ (i.e. the set of tasks needed to complete job j) as follows: $\eta(j)$ = $\{j, (j+1) \bmod q\}$. That is, each job requires two tasks and each task participates in two jobs. We take as task sequence for agent i the sequence $\mathcal{T}_i$ = $\langle (i-1) \bmod q, \ldots q-1, 0, \ldots ((i-1) \bmod q) - 1\rangle$. In other words, all agents consider tasks with the same order (namely $\langle 0, \ldots q-1\rangle$). The only difference is that agent i will start its task sequence from task $(i-1) \bmod q$. For each agent i we set, β_i = 0.5 and $\boldsymbol{\beta}$ = $\langle\, \beta_1,\, \ldots\, \beta_n\,\rangle$. With the above settings we have only two parameters to be instantiated: n (number of agents) and m (number of jobs).

C. Experimental Results

Table XI-C shows our experimental results on verification of the ε-f-Nash property for the mechanism described in Sects. XI-A, XI-B. Column *Byzantines* in Table XI-C gives the number of Byzantine agents (f). In all experiments we take $\varepsilon = 0.01$ and accuracy $\delta = 0.005$. With such settings the value of k in line 3 of Algorithm 1 turns out to be 15 in all our experiments. Column *Nash* in Table XI-C shows the result returned by Algorithm 1, namely, *PASS* if the mechanisms is ε-f-Nash or $(\varepsilon+\delta)$-f-Nash, *FAIL* otherwise. The meaning of the other columns in Table XI-C should be self-explicative.

From Table XI-C we see that we can effectively handle moderate size mechanisms. Such mechanisms correspond indeed to quite large games. In fact, given a finite horizon k, an n player mechanism can be seen as a game whose outcomes are n-tuple $\langle \sigma_1, \ldots, \sigma_n\rangle$ of strategies of length k, where σ_i is the strategy played by agent i. If the underlying behavior of agent i allows it two actions for each state, then there are 2^k strategies available for agent i. This would yield a game whose normal form has 2^{kn} entries. In the mechanism used in Table XI-C, even without considering Byzantine players, each agent can choose at least among $fib(k)$ (the k-th Fibonacci number) strategies (many more if we consider Byzantine players). With horizon $k = 15$ and $n = 8$ players this leads to a normal form game with at least $fib(k)^n$ = $610^8 \approx 10^{22}$ entries.

XII. Conclusions

We present a symbolic model checking algorithm for verification of Nash equilibria in finite state mechanisms modeling MAD distributed systems. Our experimental results show the effectiveness of the presented algorithm for moderate size mechanisms. For example, we can handle mechanisms which corresponding normal form games would have more than 10^{22} entries.

Future research work include: improvements to the presented algorithm in order to handle larger mechanisms, verification of Nash equilibria *robust* with respect to agent *collusions*.

Acknowledgments

This work has been partially supported by MIUR grant TRAMP DM24283 and by NSF grant CRS-PDOS 0509338.

References

[1] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proc. of SOSP'05*, pages 45–58. ACM Press, 2005.

[2] Anuchit Anuchitanukul and Zohar Manna. Realizability and synthesis of reactive modules. In *Proc. of CAV'94*, pages 156–168, 1994.

[3] Anish Arora, Paul C. Attie, and E. Allen Emerson. Synthesis of fault-tolerant concurrent programs. In *Symposium on Principles of Distributed Computing*, pages 173–182, 1998.

[4] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid thermal multiprocessor. *Joint issue Automatica and IEEE Trans. Autom. Control on* Meeting the Challenge of Computer Science in the Industrial Applications of Control, 1993.

[5] P. Ballarini, M. Fisher, and M. Wooldridge. Automated game analysis via probabilistic model checking: A case study. In *Proc. of MoChArt'05*, volume 149 of *ENTCS*, pages 125–137. Elsevier, 2006.

[6] Christopher Batten, Kenneth Barr, Arvind Saraf, and Stanley Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

[7] Massimo Benerecetti and Fausto Giunchiglia. Model checking security protocols using a logic of belief. In *Proc. of TACAS'00*, pages 519–534. Springer-Verlag, 2000.

TABLE I
EXPERIMENTS RUN ON A 64-BIT DUAL QUAD CORE 3 GHZ INTEL XEON LINUX PC WITH 8 GB OF RAM

Agents	Jobs	Byzantines	Nash	CPU (sec)	Mem (MB)	Max BDD	Present State Bits	Action Bits	Model Size (KB)
5	2	1	PASS	1.49e+01	7.99e+01	5.88e+05	15	5	4.99e+01
5	2	2	FAIL	2.39e+01	7.41e+01	5.42e+05	15	5	4.99e+01
6	2	2	PASS	2.09e+02	8.18e+01	6.15e+05	18	6	6.90e+01
6	2	3	FAIL	2.99e+02	8.60e+01	4.33e+05	18	6	6.90e+01
6	3	3	PASS	1.68e+03	2.77e+02	5.62e+06	24	6	1.83e+02
6	3	4	FAIL	1.14e+03	2.17e+02	5.85e+05	24	6	1.83e+02
7	3	3	PASS	1.91e+04	1.85e+03	2.29e+07	28	7	2.46e+02
7	3	4	FAIL	2.22e+04	2.28e+03	5.64e+07	28	7	2.46e+02
8	3	2	PASS	8.03e+04	4.66e+03	5.53e+07	32	8	3.18e+02
8	3	3	N/A	>1.27e+05	>8.00e+03	>3.45e+07	32	8	3.18e+02

[8] Massimo Benerecetti, Fausto Giunchiglia, Maurizio Panti, and Luca Spalazzi. A logic of belief and a model checking algorithm for security protocols. In *Proc. of FORTE'00*, pages 393–408, 2000.

[9] Rafael H. Bordini, Michael Fisher, Carmen Pardavila, Willem Visser, and Michael Wooldridge. Model checking multi-agent programs with casp. In *Proc. of CAV'03*, pages 110–113, 2003.

[10] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, Aug 1986.

[11] Levente Buttyán and Jean-Pierre Hubaux. *Security and Cooperation in Wireless Networks - Thwarting Malicious and Selfish Behavior in the Age of Ubiquitous Computing (version 1.5)*. Cambridge University Press, 2007.

[12] Zining Cao. Model checking for real-time temporal, cooperation and epistemic properties. In *Intelligent Information Processing III*, volume 228 of *IFIP International Federation for Information Processing*, 2007.

[13] Steve Chien and Alistair Sinclair. Convergence to approximate nash equilibria in congestion games. In *Proc. of SODA'07*, pages 169–178. Society for Industrial and Applied Mathematics, 2007.

[14] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[15] A. Clement, H. Li, J. Napper, J-P Martin, L. Alvisi, and M. Dahlin. BAR primer. In *Proc. of DSN'08*, June 2008.

[16] Bram Cohen. Incentives build robustness in bittorrent. In *Proc. of IPTPS'03*, Feb 2003.

[17] Landon P. Cox and Brian D. Noble. Samsara: honor among thieves in peer-to-peer storage. In *Proc. of SOSP'03*, pages 120–132. ACM, 2003.

[18] CUDD Web Page: http://vlsi.colorado.edu/∼fabio/, 2004.

[19] Constantinos Daskalakis, Aranyak Mehta, and Christos Papadimitriou. Progress in approximate nash equilibria. In *Proc. of EC'07*, pages 355–358. ACM, 2007.

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

[21] Kfir Eliaz. Fault tolerant implementation. *Review of Economic Studies*, 69(3):589–610, July 2002.

[22] E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[23] H. Everett. Recursive games. *Contributions to the theory of games, vol. III - Annals of Mathematical Studies*, 39, 1957.

[24] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *Proc. of STOC'04*, pages 604–612. ACM, 2004.

[25] Joan Feigenbaum, Rahul Sami, and Scott Shenker. Mechanism design for policy routing. In *Proc. of PODC'04*, pages 11–20, New York, NY, USA, 2004. ACM.

[26] D. Fudenberg and J. Tirole. *Game theory*. MIT Press, aug 1991.

[27] Richard D. McKelvey, Andrew M. McLennan and Theodore L. Turocy. Gambit: Software Tools for Game Theory, Version 0.2007.01.30 http://gambit.sourceforge.net/, 2007.

[28] Peter Gammie and Ron van der Meyden. Mck: Model checking the logic of knowledge. In *Proc. of CAV'04*, pages 479–483, 2004.

[29] Gear web page: http://jabc.cs.uni-dortmund.de/modelchecking/.

[30] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *LNCS*. Springer, 2002.

[31] T. A. Henzinger. Model checking game properties of multi-agent systems (abstract). In *Proc. of ICALP'98*, page 543. Springer-Verlag, 1998.

[32] I. Walukiewicz. Pushdown processes: Games and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Proc. of CAV'96*, volume 1102, pages 62–74. Springer Verlag, 1996.

[33] M. Kacprzak, A. Lomuscio, and W. Penczek. Verification of multi-agent systems via unbounded model checking. In *Proc. of AAMAS'04*, July 2004.

[34] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[35] M. Leucker. Model checking games for the alternation free μ-calculus and alternating automata. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of LPAR'99*, volume 1705 of *LNAI*, pages 77–91. Springer, 1999.

[36] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proc. of OSDI'06*, November 2006.

[37] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Mike Burrows, and Michael Isard. A cooperative internet backup scheme. In *Proc. of ATEC'03*, pages 29–41. USENIX Association, 2003.

[38] Alessio Lomuscio and Franco Raimondi. Model checking knowledge, strategies, and games in multi-agent systems. In *Proc. of AAMAS'06*, pages 161–168. ACM, 2006.

[39] C. Stirling M. Lange. Model checking games for branching time logics. *Journal of Logic and Computation*, 12:623–639, 2002.

[40] Ratul Mahajan, Maya Rodrig, David Wetherall, and John Zahorjan. Sustaining cooperation in multi-hop wireless networks. In *Proc. of NSDI'05*, pages 231–244. USENIX Association, 2005.

[41] Petros Maniatis, David S. H. Rosenthal, Mema Roussopoulos, Mary Baker, TJ Giuli, and Yanto Muliadi. Preserving peer replicas by rate-limited sampled voting. In *Proc. of SOSP'03*, pages 44–59. ACM, 2003.

[42] F. Mari, I. Melatti, I. Salvo, E. Tronci, L. Alvisi, A. Clement, and H. Li. Model checking nash equilibria in mad distributed systems. Technical report, Computer Science Department, University of Rome "La Sapienza", http://modelcheckinglab.di.uniroma1.it/techreps/bar-techrep-2008-07-24.pdf, 2008.

[43] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proc. of SP '97*, page 141. IEEE Computer Society, 1997.

[44] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Seventh USENIX Security Symposium*, pages 201–216. USENIX, San Antonio, 1998.

[45] Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proc. of STOC'99*, pages 129–140. ACM, 1999.

[46] NuSMV Web Page: http://nusmv.irst.itc.it/, 2006.

[47] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1):81–98, 1989.

[48] Fred B. Schneider. *What good are models and what models are good?* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[49] Jeffrey Shneidman and David C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. of PODC'04*, pages 88–97. ACM, 2004.

[50] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Symposium on Theoretical Aspects of Computer Science*, pages 1–13, 1995.

[51] Wolfgang Thomas. Infinite games and verification (extended abstract of a tutorial). In *Proc. of CAV'02*, pages 58–64, 2002.

[52] Enrico Tronci. Optimal finite state supervisory control. In *Proc. of CDC'96*. IEEE, 1996.

[53] Enrico Tronci. Automatic synthesis of controllers from formal specifications. In *Proc. of ICFEM'98*, page 134. IEEE, 1998.