Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 203 (2008) 53–67
www.elsevier.com/locate/entcs

Validating for Liveness in Hidden Adversary Systems

Saikat Mukherjee[1] Srinath Srinivasa[2] SatishChandra D[3]

International Institute of Information Technology, Bangalore, India 560100

Abstract

Multi-stream interactive systems can be seen as "hidden adversary" systems (HAS), where the observable behaviour on any interaction channel is affected by interactions happening on other channels. One way of modelling HAS is in the form of a multi-process I/O automata, where each interacting process appears as a token in a shared state space. Constraints in the state space specify how the dynamics of one process affects other processes. We define the "liveness criterion" of each process as the end objective to be achieved by the process. The problem now for each process is to achieve this objective in the face of unforeseen interferences from other processes. In an earlier paper, it was proposed that this uncertainty can be mitigated by *collaboration* among the disparate processes. Two types of collaboration philosophies were also suggested: *altruistic* collaboration and *pragmatic* collaboration. This paper addresses the HAS validation problem where processes collaborate altruistically.

Keywords: Validation, open-world systems, hidden-adversary systems, collaboration, altruistic collaboration.

1 Introduction

Information systems can be broadly classified into three kinds: algorithmic systems, hidden-variable systems and hidden-adversary systems (HAS) [5,11,9].

Figure 1(a) depicts an algorithmic system. A purely algorithmic system represents a *closed world* computation with exactly one input and one output interaction. Mathematical and library functions in application programs are examples of algorithmic processes. Their observable behaviour is in the form of an atomic transition from the start of the computation to its end.

Figure 1(b) shows a hidden-variable system, also called a single-stream interactive machine (SIM). Here, the observable behaviour of the system for a given

[1] Email:saikat.mukherjee@iiitb.ac.in
[2] Email:sri@iiitb.ac.in
[3] Email:satishchandra.d@iiitb.ac.in

1571-0661 © 2008 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2008.04.086

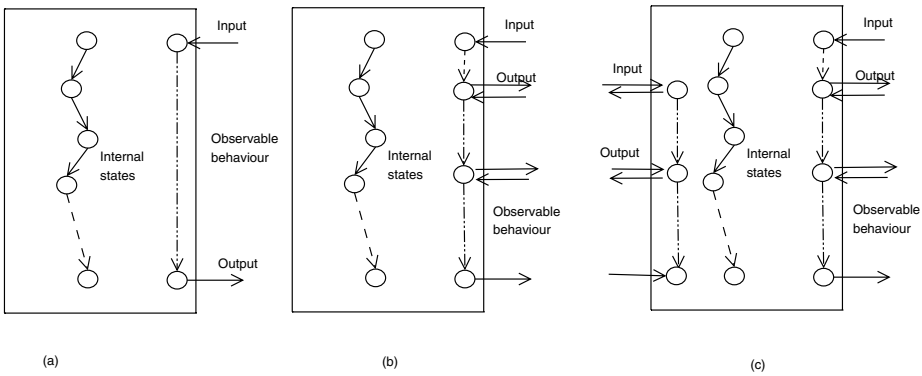

Fig. 1. Algorithmic, Hidden Variable and Hidden Adversary Systems

input is dependent not only on the current input values, but also on the *history* of previous interactions.

Such systems can be modelled as *hidden-variable* systems, where one or more unobservable variables encapsulate the history of previous interactions. The interactive process may be of finite duration after which, the process ends and the history of interactions is discarded. On the other hand, the interactive process may be of infinite duration, where the history is never discarded.

Most information systems are of a third variety as depicted in Figure 1(c). The system interacts over *multiple* interaction streams simultaneously. The streams need not be independent of one another and the interactive dynamics on one stream can interfere with the observable behaviour on another stream. This interference may be by design or unintended.

The observable behaviour of a process on any interaction channel cannot be adequately modeled by hidden variables. The processes would seem to be affected by hidden *adversaries* that influence their observable behaviour. We call such systems as hidden-adversary systems (HAS).

Interactive processes in a HAS could be of finite or infinite duration. In this work, we will only be considering finite duration processes. This means that the interactive behaviour on each channel can be modelled as "sessions" of a given duration.

One of the main challenges in designing multi-stream interactive systems is to model the space of these interferences and manage them effectively at run-time. Unfortunately, most of the existing meta-models for system design, focus primarily on the problem of modelling interaction that is part of the service logic.

Interferences in a HAS are not part of the service logic. As a result, they may not be sufficiently accounted for during system specification and design. They arise because of the shared state space used by all the processes. Interferences by hidden adversaries cause individual processes appear to behave erratically or very inefficiently, even though in isolation, they appear to work perfectly. It is hence imperative for the system designer to adequately model interferences in the system and provide guarantees of liveness and safety in the presence of hidden adversaries.

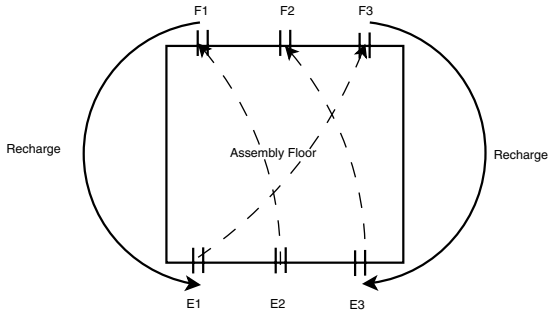

Fig. 2. Example Assembly Line Scenario

In this paper, we address a specific part of liveness checking for HAS. Since the interactive process on each channel is in the form of a finite-duration session, we define the liveness criterion as being able to successfully complete a session in the face of unexpected interferences from other processes.

In an earlier paper [10], we had suggested that individual processes in a HAS, can mitigate uncertainty by *collaborating* with other processes in the system. The application scenario we have in mind is of systems made of multiple autonomous agents, each of which are pursuing their own individual objectives. Since they share the same state space, they interfere with one another's progress.

For instance, consider the assembly floor example shown in Figure 2, where $E1$, $E2$, $E3$ are the different entry points for autonomous assembly line robots which collectively help build a product and exit via $F3$, $F1$ and $F2$ respectively. The liveness criterion for each robot is to finish its allocated work and exit the assembly floor to recharge itself. However, because of various dependencies and interferences, a robot would not be able to unilaterally reach its end state. The robots should collaborate and develop a collective strategy to mitigate uncertainties that arise because of their mutual interferences.

A concept called "collaboration number" was also introduced in [10] that can be used to calibrate a HAS system model by specifying the minimum number of collaborators required for processes to achieve their liveness criteria. The term collaboration number actually refers to a set of numbers defining different parameters pertaining to collaboration. In this paper, we expand on this work and introduce algorithms to identify the collaboration number of a class of hidden-adversary systems.

2 Interaction Schema

Before addressing the validation problem for HAS, we introduce a meta-model to capture interferences. The meta-model interaction schema [4] was first introduced in [9], but has subsequently undergone several changes. Essential details of the interaction schema is provided here for the sake of completeness.

[4] Strictly speaking, it should be called "interference schema" rather than "interaction schema"; but we shall be using the latter term, since it is already in use.

Formally, an interaction schema is an I/O automaton [7] having a shared state space and augmented with normative constraints. It is of the form:

$$\mathbb{I} = (S, L, S_0, H, \delta, \psi)$$

where S is a set of states that is shared by all processes using the space; L is a set of labels describing tasks (input, output, internal actions) performed by processes; $S_0 \subset S$ is a set of start states, where processes are created; and $H \subset S$ is a set of end states, where processes seek to end up in. $\delta : S \times L \rightarrow S$ is a transition function that defines state transitions by processes on performing tasks in L. $\psi \subseteq S \times M \times S$ is a set of constraints that describe interferences among processes in the space. The term M refers to a constraint *modality*. In this model $M = \{O, P, F\}$ indicating obligated, permitted and forbidden modalities respectively.

Any constraint $c \in \psi$ is written in the form $s \rightarrow M[s']$ and is read as: *if there is a process in state s, then state s' gets a modality M*. Interpretation of the modalities are as follows:

- $s \rightarrow P[s']$. If there is a process in state s, only then is it permitted for a process to *enter* state s'
- $s \rightarrow F[s']$ As long as there is at least one process in state s, it is forbidden for any process to *enter* state s'
- $s \rightarrow O[s']$ If there is a process in state s, it is obligated that another process should be in state s'. In other words, as long as there is no process in state s', the process in s cannot *leave* state s.

We can note that the constraints P and F act on *entry* conditions, while O which is a waiting constraint, acts on the *exit* condition of a state. The O constraint also acts in the reverse direction, constraining the tail-end of the rule, rather than the head.

A process can be created in any $s_0 \in S_0$ only if the modality of s_0 is P, and any process can exit the system from any $h \in H$ only if there are no unsatisfied O constraints from h. When a state has two or more active incoming constraints, the net modality that it obtains is the conjunction of all the incoming modalities. Conjunction rules are defined as follows: $O \wedge O \Rightarrow O$, $P \wedge P \Rightarrow P$, $F \wedge F \Rightarrow F$, $P \wedge F \Rightarrow F$, $O \wedge P \Rightarrow O$, $O \wedge F \Rightarrow F$

In addition to the above properties, the interaction schema also has the following properties:

- *Constraints apply to processes other than the one which poses the constraint.* Some typical examples are given below:

(i) A constraint of the form $s \rightarrow F[s]$ denotes a critical section, where only one process can be in state s at any time.

(ii) A constraint of the form $s \rightarrow O[s]$ denotes a livelock, where once a process enters state s, it cannot leave the state until another process enters s.

(iii) The self-constraint $s \rightarrow P[s]$ denotes an implausible condition.

- *Transitions between states are assumed to be atomic and the constraint is assumed to hold till the transition is complete.* Hence if there is a constraint of the form

$s \rightarrow P[s']$, a process that has entered state s can now also move to enter state s' if there is a transition from s to s'. The permission on s' ceases to hold only after the process has successfully entered the next state after s.

- *When a constraint ceases to hold, the negation of the constraint is now said to hold.* The negation rules for constraints are as shown: $\neg P \Rightarrow F$, $\neg F \Rightarrow P$, $\neg O \Rightarrow P$

In general the system is assumed to be *maximal*; that is, any state with no incoming constraint is said to have a P modality by default. (Everything is permitted, unless forbidden). Only those states that have an incoming P constraint are minimal. They are forbidden, unless their incoming P constraints are active.

In practical applications, the interaction schema is generalized to include richer constraint semantics. (See for example, LogicFence [6], which is an implementation of interaction schema). However, as the objective here is to ultimately develop a theory for HAS, we start with simple formulations of the interaction schema model. We are also aware of several limitations about the expressiveness of the current model of the interaction schema. For instance, constraint disjunctions are not defined. Similarly, constraints do not propagate in the system. However, as a first step towards developing a theory of HAS, we will be working with these simplifications in this paper.

3 The Collaboration Number of an Interaction Schema

Given an interaction schema $\mathbb{I}$ and a process x, the process begins execution in some state $s_0 \in S_0$ (if it is permitted), and performs a series of tasks to finally reach one of the end states $h \in H$. It then exits the system if there is no unsatisfied outgoing O constraint from h.

Because the journey from the start to the end state is fraught with uncertain interferences from other processes, a process seeks to mitigate these uncertainties by *collaborating* with other processes, whose behaviours it can reasonably predict. Other than the collaborators, all other processes in the system are said to be *adversaries*, which can potentially block the process from reaching its end state.

For the purposes of validation, we take a worst-case scenario where the objective of any adversary is to *block* all paths to end states, while the objective of the collaborators is to *open* as many paths to the end state as possible, by blocking the adversaries' plans.

Definition 3.1 Given an interaction schema $\mathbb{I}$, and a path $\mathbb{S} = s_0 \ldots h$ in the state space of $\mathbb{I}$ from the start state to one of the end states, a *collaboration number* $L_k^{m,n}(\mathbb{S})$ is defined for path $\mathbb{S}$ in $\mathbb{I}$ where: k is the minimum number of collaborator processes required to guarantee that no adversary can block any state in $\mathbb{S}$, and for every m steps taken by the adversaries, the set of collaborators can take n steps. (3.1)

In the above definition, a "step" is defined as one of the following:

(i) Creation of a process

(ii) A process executing a state transition

(iii) A process exiting the system

There are these important issues to note in the above definition:

- A system with a collaboration number $L_k^{m,n}(\mathbb{S})$ is not reducible to a system of the form $L_k^{1,n}(\mathbb{S})$. For instance consider a system, where, for every two steps of the adversary, collaborator processes take three steps.
- A system where the adversaries' processes have a head start of m steps is defined as a $-L_k^{m,n}(\mathbb{S})$ system. However this paper does not address systems where adversaries get a head start. In fact, in this work, we assume that all the collaborator process move into their respective positions even before the adversaries take a single step. For such systems, the collaboration number will be of the form $L_k^{1,n}(\mathbb{S})$.

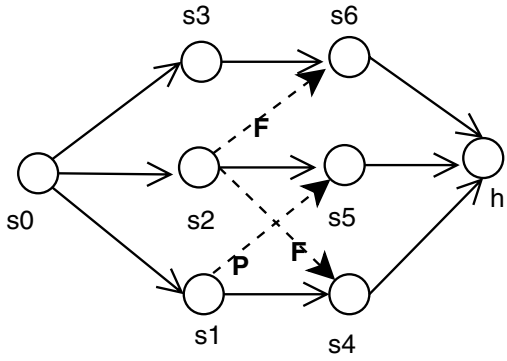

Fig. 3. An $L_1^{1,1}(s_0s_2s_5h)$ system

Figure 3 shows a system that is $L_1^{1,1}$ when the path $s_0s_2s_5h$ is considered. With one collaborator process entering state s_1, this path is freed from any incoming blocking constraints. As long as a collaborator process is in s_1 any number of processes can traverse $s_0s_2s_5h$ and satisfy their liveness criteria.

For the paths $s_0s_3s_6h$ and $s_0s_1s_4h$, the collaboration number is said to be L_∞. No matter how many collaborator processes are there, these paths can never be freed of adverse interferences, since an adversary entering state s_2 can block both these paths.

In fact, in Figure 3, the collaborator process for the path $s_0s_2s_5h$ does not have its own liveness criteria guaranteed, because its path $s_0s_1s_4h$ always has an uncertainty. So any collaborator in this system wishing to free the path $s_0s_2s_5h$ needs to "sacrifice" its own liveness guarantees for the sake of the path.

This brings us to define two kinds of collaboration: *altruistic* and *pragmatic* collaboration. The term L in Definition 3.1 is hence replaced by either A or P to denote the collaboration type.

Definition 3.2 In an interaction schema $\mathbb{I}$, a given path $\mathbb{S} = s_0 \dots h$ from one of the start states to one of the end states, is said to have an *altruistic* collaboration number $A_k^{m,n}(\mathbb{S})$ if there exists a set of k processes and k states such that:

- There exists a sequence of steps where the k processes can occupy these k states, where for every n steps taken by the set of collaborators, adversaries can take m steps
- After the k states have been occupied, the path $\mathbb{S}$ is free of uncertainty as long as the k states are occupied
- The k collaborator processes have no guarantees regarding their own liveness criteria. (3.2)

Based on this definition, we can see that the interaction schema of Fig 3 has an altruistic collaboration number $A_1^{1,1}(s_0 s_2 s_5 h)$.

Alternatively, we can also think of an interaction schema where collaboration is symbiotic, that is, a set of collaborating processes mutually mitigate uncertainties.

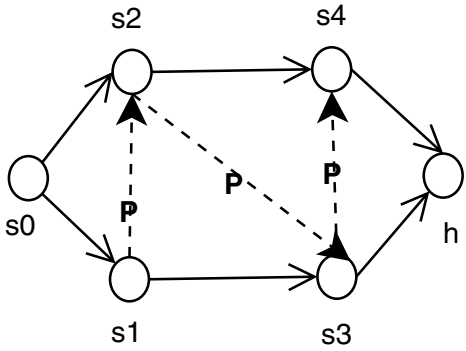

Fig. 4. A system allowing for pragmatic collaboration: A $P_1^{1,1}(s_0 s_2 s_4 h, s_0 s_1 s_3 h)$ system

In Figure 4, we can see that a pair of processes can mutually help each other to reach the end state. Given a process p_1 and a collaborator process p_2, p_2 first goes to state s_1 and waits for the p_1 to enter s_2. Next, p_2 enters state s_3 and waits for p_1 to enter s_4. Following this, both can enter the final state h. This kind of collaboration is termed *pragmatic* collaboration.

We shall not be going into details of pragmatic collaboration and shall restrict ourselves to altruistic collaboration checking in this paper.

Incidentally, Figure 4 also has an altruistic collaboration number $A_2^{1,1}(s_0 s_1 s_3 h)$ and $A_3^{1,1}(s_0 s_2 s_4 h)$. For the former path, there needs to be two collaborators occupying states s_1 and s_2; while for the latter path, there needs to be three collaborators occupying states s_1, s_2 and s_3. There are however, examples of interaction schema that allow pragmatic collaboration, but have no altruistic collaboration number. This is beyond the scope of this paper.

In this paper, we shall also be restricting ourselves to interaction schema that have a single start state s_0. Given an interaction schema having multiple start states (denoted by set S_0), we reduce it to one having a single start state for the

purposes of validation. This is done by creating a single start state s_0 and adding ϵ-transitions from s_0 to each of the states in S_0.

This is not an entirely accurate translation, since it affects the m and n counts in the collaboration number. Also, if an incoming F constraint holds on a start state $s \in S_0$, in the former case, no process can be created at all, while in the latter case, a process can be created in s_0 where it is stuck while trying to enter s. However, as we will see in the next section, these inaccuracies do not affect validating of altrusitic collaboration.

4 Liveness Validation for Altruistic Collaboration

As described earlier, the altruistic collaboration number of a given schema is of the form $A_k^{m,n}(\mathbb{S})$; where k is the number of collaborative processes required to guarantee that $\mathbb{S}$ is a *safe path* (i.e. a fully permitted path). Also m is the number of steps the adversary processes can take for n steps taken by the collaborator processes.

In the simplest form of altruistic collaboration that we will be addressing here, the following assumption holds: all collaborator processes take all their required steps to ensure a safe path, before the adversary processes take even a single step. The collaboration number for a system with such a simplifying assumption, takes the form, $A_k^{1,j}(\mathbb{S})$.

Even with such a simplification, finding the collaboration number for such a system is non-trivial because given any path, it involves the following steps:

(i) Determining the number of collaborator processes k required to ensure a safe path and the states the processes need to be in (also called as *dependent states*).

(ii) Calculating the total number of steps j, the set of k collaborators need to take to move into their designated states.

(iii) Providing a sequence of moves which will ensure that all the k collaborators can move into their designated states (in the presence of constraints between the dependent states).

The optimal solution to the problem would be to determine the *minimum* number of collaborator processes required (k_{min}) and the *minimum* number of steps the collaborator processes need to take (j_{min}), along with the sequence of steps. However, finding the optimal solution turns out to be NP-hard. Hence we provide heuristics in the following section to determine some non-trivial, safe values for the number of collaborators and the number of steps they have to take.

In order to develop the heuristics, we define the following terms for an interaction schema $\mathbb{I} = (S, L, s_0, H, \delta, \psi)$:

Clean state: Any state $s \in S$ is called a clean state if its modality is always P. This means that there is no incoming constraint on s, or it can be shown that even with the incoming constraint, the resultant modality on s will be P. (4.1)

Safe path: Any path between any two states in S is said to be a safe path, if it made up of all clean states. (4.2)

Dependent state: Any state that a collaborator process needs to occupy in order to ensure a safe path for the required liveness criteria is called a dependent state. (4.3)

In addition, we also make an implicit assumption that if all constraints were to be removed from the schema, all states $s \in S$ are reachable from the start state s_0.

4.1 Heuristic H1: Number of collaborator processes (k)

In this heuristic, we concentrate on the problem of determining the number of altruistic collaborators k required in order to ensure that a given path $\mathbb{S}$ is a safe path. Later on, we will look into determining the number of steps j required for collaborators to enter their respective states and the sequence in which the states have to be occupied. We begin this heuristic with the following lemmas.

Lemma 4.1 *Before any adversary enters the system, a process requires collaborators only to enter states that have incoming P constraints.*

Proof. For a given state s, the only constraints that affect its entry are incoming P and F constraints. The O constraint only acts on the exit of processes from states and hence is not relevant here.

Since there are no adversaries in the system, a process can by default enter a state that has an incoming F constraint without any collaborators.

However, a state having an incoming P constraint is forbidden unless the constraint is active. A process cannot enter such a state without having a collaborator to trigger the P constraint. □

Lemma 4.2 *In an altruistic system of the form $A_k^{1,j}(\mathbb{S})$, any state $s \in \mathbb{S}$ that has an incoming chain of 1 or more P constraints of the form $s_i \rightarrow P[s_{i+1}], s_{i+1} \rightarrow \ldots \rightarrow P[s]$, can be made into a clean state only if s_i is clean.*

Proof. This can be proved by induction. Assume that there is a chain of n P constraints leading to state s. When $n = 1$, it constitutes a constraint of the form $s_i \rightarrow P[s]$. In such a case, it is apparent that state s can be made clean by a collaborator process entering state s_i. This can happen only if s_i is clean.

Assume now that the lemma holds for an arbitrary value of n. That is, there is a set of n P constraints starting from state s_i leading to state s, and state s can be made clean only if state s_i is clean.

Since we assume that all collaborators take their positions before the adversaries even take their first step, the only way state s_i can be unclean is by an incoming P constraint (Lemma 4.1). Adding a P constraint to state s_i from another state s_j increases the length of the chain by 1, and it follows from the previous argument that state s_i and hence state s, can be made clean only if state s_j is clean. □

Note that as a result of the above lemma, cycles arising because of a set of constraints of the form $s \rightarrow P[s']$ and $s' \rightarrow P[s]$ can be eliminated from consideration, as both s and s' can never be assured to be clean.

Lemma 4.3 *Any state s that has an incoming FP^*F constraint chain, can be made clean with just one collaborator.*

Here, FP^*F means an F constraint acting on s from a state on which there is a chain of 0 or more P constraints, that ends with an F constraint.

Proof. If the constraint leading to s is of the form $s_j \rightarrow F[s]$, then s would be unclean if s_j is clean (allowing an adversary to enter it at any time).

Since there is a chain of P constraints acting on s_j, it follows from Lemma 4.2 that s_j can be clean only if the state s_i at the head of the P chain is clean.

Since there is an incoming F constraint of the form $s' \rightarrow F[s_i]$ leading to the head state of the P chain, a collaborator process entering s' can ensure that s_i is never clean. □

Based on these lemmas, we can now determine whether $k = \infty$ as follows:

Theorem 4.4 *An interaction schema is an $A^{1,j}_{\infty}(\mathbb{S})$ system only if there exists a state $s \in \mathbb{S}$ which has an incoming chain of FP^* or FO constraints or there exists a constraint of the form $s' \rightarrow P[s]$, where $s, s' \in \mathbb{S}$ and s' can only be reached via s.*

Here FP^* means an F constraint acting on s of the form $s' \rightarrow F[s]$, and on s', there is a chain of 0 or more P constraints of the form $s_i \rightarrow P[s_{i+1}], s_{i+1} \rightarrow \ldots \rightarrow P[s']$. The term FO means a pair of constraints of the form $s' \rightarrow F[s]$ and $s' \rightarrow O[s'']$, that is, an incoming F constraint into s and an outgoing O constraint from the state forbidding s.

Proof. In order to prove this, we first note that there should be at least one state $s \in \mathbb{S}$ that has an incoming P or F constraint or an outgoing O constraint, in order for k (the number of required collaborators) to be greater than 0. If we can show that the constraints leading to s are such that, we can never assure s to be clean, then we see that $k = \infty$.

If there is a P constraint of the form $s' \rightarrow P[s]$, where $s, s' \in \mathbb{S}$, to enter state s, there must be a collaborator process in state s'. Since the only path to s is via s and s cannot be entered, the system is an $A^{1,j}_{\infty}(\mathbb{S})$ system.

If there is a chain of one or more P constraints leading to s, from Lemma 4.2, s can be made clean by collaborators entering the chain. It does not matter if the P chain has incoming F or O constraints from anywhere since we assume that there are no adversaries in the system until all collaborators have taken their positions (Lemma 4.1).

If there is an O constraint of the form $s \rightarrow O[s']$ leading from s, we can clean s, by placing a collaborator in s'. This requires us to ensure that s' is clean, which can be assured trivially if there are no constraints on s' or there are only incoming F constraints. An incoming P chain of constraints can be handled as shown in Lemma 4.2. An outgoing O constraint from s' will be a recursive formulation of the same problem.

The only way now that s can be left unclean is to have a constraint of the form $s' \rightarrow F[s]$, where s' is assured clean (thus allowing an adversary to enter s' at any time). As shown in the previous steps, s' can be clean if:

(i) No constraint is acting on s', or

(ii) There is a chain of P constraints starting from a clean state, acting on s', or

(iii) There is an outgoing O constraint from s' (which does not prevent entry into s')

This gives the FP^* and FO chains specified in the theorem. □

Figure 5 shows fragments of an interaction schema depicting different cases when k is ∞.

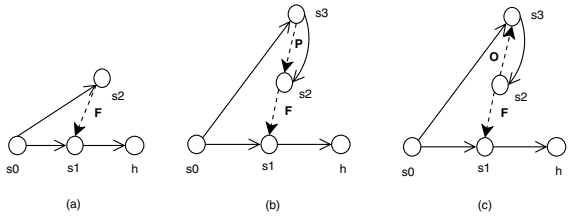

Fig. 5. A_∞ systems

Figure 6 illustrates the FP^*F chain described in Lemma 4.3, where a state can be made clean with just one collaborator process.

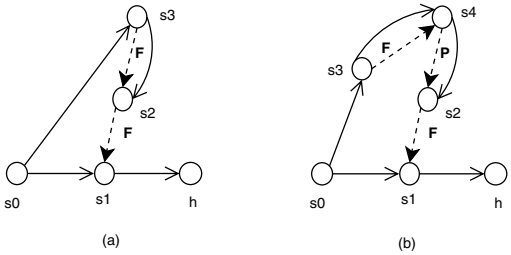

Fig. 6. Systems with F rules

4.2 Algorithm A1: Number of Collaborator Processes

Given these underpinnings, we now turn to the problem of determining k for an arbitrary interaction schema. To determine the number of collaborator processes k required, we perform the following steps:

(i) **Initialization:** We define two sets, a *dependency set* D and a *collaboration set* C which are initially empty. $D = \{\}$, $C = \{\}$

(ii) **Determine if the system is an $A_\infty^{1,j}(\mathbb{S})$ system:** For each state $s \in \mathbb{S}$, we check if it has an incoming FP^* chain of constraints or an FO chain or a P constraint of the form $s' \rightarrow P[s]$, where $s, s' \in \mathbb{S}$ and s' can only be reached via s. If there is such a state, then the system is an $A_\infty^{1,j}(\mathbb{S})$ system (Theorem 4.4).

(iii) **Else, Determine the Dependent states:** For each state $s \in \mathbb{S}$, we check if it has an incoming FP^*F or P constraint or an outgoing O constraint. We add all such states in the dependency set D.

(iv) **Paths to Dependent states:** For each state $s \in D$, if it is not in C, we determine a path p from the start state s_0 to the state s. There might be multiple such paths, which we store in a stack (for backtracking later, if required) and select a path p_{min} that has the minimum number of states with incoming P constraints. We do this to reduce the number of collaborator processes required. Only incoming P constraints are considered because of Lemma 4.1. All states in path p_{min} with an incoming P constraint are added to the dependency set D. We then add state s to the collaboration set C, provided state s does not have a cyclic dependency of F constraints of the form, $s \rightarrow F[s']$ and $s' \rightarrow F[s]$ with another state $s' \in C$. In the event of such a dependency, we backtrack to another path.

(v) **Backtracking condition:** The algorithm backtracks to another path in the stack when there is no state $s \in D$ which can be added to C and $D - C \neq \{\}$

(vi) **Termination condition:** The algorithm terminates when: $\forall s \in D \Rightarrow s \in C$

The cardinality of the collaboration set C gives the number of collaborator processes k required.

4.3 Heuristic H2: Total number of initial steps (j)

The total number of initial steps j required to place the k collaborator processes in the respective states is given by:

$$j = k + \sum_{i=0}^{k} numStates(p_{min}(i))$$

where $numStates(p_{min}(i))$ returns the number of states in path $p_{min}(i)$, where $p_{min}(i)$ is the p_{min} path generated in step 4 of Algorithm A1 for the i^{th} state in C. The factor k is added because process creation is also considered to be a step.

4.4 Heuristic H3: Sequence of initial moves

Once the dependent states are known for ensuring liveness along a particular path, the k processes need to be put in the dependent states. However, there may be constraints between the dependent states and therefore any arbitrary sequence of moves to place the processes in the dependent states may not work. For instance in the example shown in Figure 7(a), collaborator processes need to be placed in states s_2, s_3, s_5 and s_6. However, it is apparent that s_5 is the first state that needs to be occupied by a collaborator, before other states.

In order to resolve such dependencies, we introduce the notion of *priority states*.

Definition: If a constraint exists between any two states, $s_x, s_y \in D$, the *priority state* is the state (either s_x or s_y) that needs to be occupied first by the collaborator. (4.4)

Table 1 indicates the priority state for all the three modalities. Here, we assume

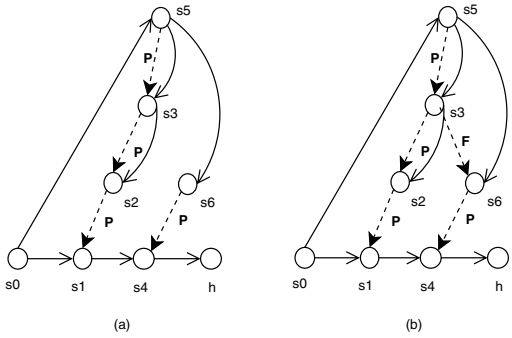

Fig. 7. Dependency Sequence Examples

Table 1
Priority States

Modality (M)	Priority State
P	s_x
F	s_y
O	–

that the constraint is of the form: $s_x \rightarrow M[s_y]$. In case of an obligated constraint, the order in which the processes are placed in s_x and s_y does not matter.

Once we have established the priority states, a *partial ordering* is created for D to determine the order in which the processes should be placed in the dependency states. The partial ordering for the example in Figure 7(a) is given in Figure 8(a) and for the example in Figure 7(b) is given in Figure 8(b).

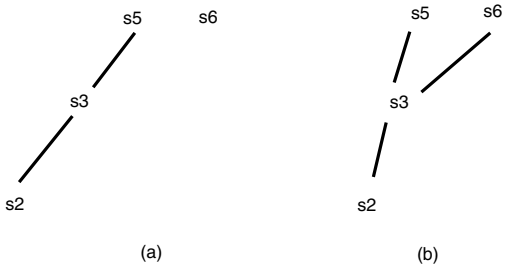

Fig. 8. Partial Ordering

5 Related Work

Several kinds of coordination models have been proposed in literature. In [8], we categorize coordination models into different classes: Connectors (ex: Petri nets), Normative (ex: contracts), Data spaces (ex. Linda), etc.

However, there are fundamental points of departure between interaction schema and coordination models. Coordination models, describe interaction and depen-

dencies that are part of the service logic of the system. Interaction schema on the other hand seeks to model *interference* among disparate mutually-autonomous processes pursuing their own goals. There need not be any larger shared objective to be collectively achieved. Collaboration features only as an *emergent* property of convenience, that is used by processes to reduce uncertainties in the system, rather being part of the service logic. In an interaction schema, the actual steps of coordinated activity that a set of agents may eventually execute, might well have been unimaginable by the system designer.

Interaction schema can be seen as a natural complement to normative multi-agent systems [1,2]. A significant amount of work has gone into normative reasoning in multi-agent systems. Interaction schema however, concentrates on constraints that are part of the agent *space* rather than part of the agents' reasoning itself. The concept of electronic institutions [4], that define a formalism to design and analyze inter-agent protocols is somewhat similar. While the design principles of electronic institutions are oriented for application in multi-agent systems, interaction schema is primarily targeted towards developing a theory for HAS with the simplest possible constructs.

While traditional model-checking using finite state machine(FSM) models [3] check if the required specifications are satisfied using only state transitions, the interaction schema, additionally considers the constraints between states while checking a given model. There have been analogies drawn between interaction schema and models like security automata and execution monitors. For a comparison of LogicFence, a practical implementation of interaction schema with these models, the reader is directed to Guha, et. al [6].

6 Conclusions and Future Work

This paper addressed the validation problem on a simplified HAS model for altruistic collaboration. Perhaps the biggest shortcoming of the current approach is the assumption that no adversaries enter the system until the collaborators take their positions. Essentially this addresses the question of how to close an open-world system. However, even when altruistic collaboration is considered, it is necessary to address situations where collaboration happens in the face of adversarial influence. Since the adversaries and their plans are hidden, collaboration in such situations would need to take the form of (game theoretic) strategies with probablistic plans. As an extension to this work, we plan to explore the integration of system design that addresses a system-wide teleological objective, with game theoretic strategies that address individual process objectives.

References

[1] G. Boella, Joris Hulstijn, and L. van der Torre. Coordination in normative multiagent systems. In *Proceedings of FINCO'05*, 2005.

[2] G. Boella, L. van der Torre, and H. Verhagen. Introduction to normative multiagent systems. In *Proceedings of AISB*, 2005.

[3] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. In *ACM Transactions on Programming Languages and Systems*, volume 8, pages 244–263, 1986.

[4] Marc Esteva, Wamberto Vasconcelos, Carles Sierra, and Juan A. Rodrguez-Aguilar. Norm consistency in electronic institutions. In *Lecture Notes in Artificial Intelligence*, volume 3171, pages 494–505, 2004.

[5] Dina Goldin, Srinath Srinivasa, and Bernhard Thalheim. Information systems = databases + interaction. In *Proceedings of ER 2000*, Salt Lake City, Utah, USA, October 2000.

[6] Shibashis Guha, Srinath Srinivasa, Saikat Mukherjee, and Ranajoy Malakar. Logicfence: A framework for enforce dynamic integrity constraints and run-time. In *Proceedings of IDEAS'06*, December 2006.

[7] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[8] Saikat Mukherjee and Srinath Srinivasa. Issues in logicfence: A symmetric contractual coordination framework. In *Proceedings of the Third Workshop on Software Design and Architecture (SoDA)*, Bangalore, India, December 2004.

[9] Srinath Srinivasa. *An algebra of fixpoints for characterizing interactive behavior of information systems*. PhD thesis, Brandenburg Technical University at Cottbus, Germany, 2001.

[10] Srinath Srinivasa and Saikat Mukherjee. Towards a validation meta-model for hidden-adversary systems: A position paper. In *Workshop on Evolutionary Models of Collaboration (EMC'07)*, Hyderanad, India, January 2007. IJCAI Workshop Proceedings.

[11] Peter Wegner and Dina Goldin. Interaction as a framework for modeling. In *Chen, et. al. (Eds.) Conceptual Modeling: Current Issues and Future Directions, LNCS 1565*, April 1999.