Detecting Coordination Failures by Observing Groups: A Formal Approach

Michael Lindner and **Meir Kalech** and **Gal A. Kaminka**
The MAVERICK Group
Computer Science Department
Bar Ilan University, Israel
{kalechm, galk}@cs.biu.ac.il

Abstract

Teams often require their members to coordinate with respect to specific aspects of their joint task. These coordination relations can often fail as a result of intermittent failures of sensor readings, communication failures, etc. Detection of such failures, based upon agents' observations of each other, is of prime importance. Though different solutions have been presented, none has presented a comprehensive, formal approach to solving this problem. Our research has produced a compact matrix-based representation of pre-defined agent coordination relations, that is used with a matrix-based representation of a plan-recognition process. The result is a novel solution that is both generic and efficient for large-scale teams. Additionally, it facilitates easy design of coordination requirements, modularity and reuse of existing systems.

1 Introduction

Autonomous agents within multi-agent systems interact and coordinate between themselves in order to achieve their goals. The increased deployment of robotic and agent teams in complex dynamic settings has, in turn, led to an increasing need for responding to coordination-failure [Kaminka and Tambe, 2000; Parker, 1998] Detection of coordination failures is essential for later recovery process during which cooperation is reinstated (e.g., by negotiations [Kraus *et al.*, 1998]). Coordination-failure detection does not indicate whether the group is achieving its goals but only if agent-coordination exists.

The following example demonstrates the importance of this process [Kaminka and Bowling, 2002]. A helicopter squadron is sent on a strike mission. According to the flight plan, at certain coordinates, one helicopter is to proceed and get visual target confirmation, while the rest of the squadron remains hidden. Once the target is confirmed the lead helicopter is to signal the squadron to commence their attack. Coordination difficulties, in this example, may lead to the failure of the mission. Thus, if an additional helicopter continues flying, it may jeopardize the whole mission. If, on the other hand, the errant pilot is able to recognize his error on time, the mission might be saved.

The systematic detection method we have devised infers the agents' internal state through observation of the agent. Some previous coordination-failure detection methods have assumed that the agents internal state is known [Klein and Dellarocas, 1999]. These methods have ignored scale-up challenges. Some have take a certain amount of uncertainty into account, but were unfortunately only able to capture specific coordination failures, such as disagreements over a selected joint plan [Kaminka and Tambe, 2000; Kaminka and Bowling, 2002]. None of the methods introduced so far has taken a systematic approach to addressing this challenge (see Section 2).

We suggest a new compact way to represent (1) pre-defined agent coordination models and (2) the agents states as inferred from their observed actions. Accounting for both the coordination model and the observation, we suggest an efficient failure-detection algorithm. The algorithm does, in many cases, reduce the complexity of detection in large-scale teams from exponential to polynomial.

The paper is organized as follows. Section 2 presents related work. Section 3 motivates the research. Then, in section 4, we present our new notation to the coordination-definition and observation modeling, and give the fault detection algorithm. An extension of the former for complex systems is presented in section 5. At last, section 6 summarizes.

2 Related Work

[Horling and Lesser, 1999; Horling *et al.*, 2001] presented a framework for diagnosing failures in multi-agent systems, based on agent information-sharing, and a diagnosis causal model. However, this work addresses neither the scale-up issues, nor the construction of the causal model that enables it to detect and diagnose failures.

A different approach was presented by [Klein and Dellarocas, 1999] according to which, each agent is paired with a sentinel. Sentinels report agent-activities to a failure-detection system that utilizes a pre-analyzed coordination failure database. This method, the failure-model approach, dictates that all possible failures be analyzed in advance. No allowance is given to different agent-action interpretation, e.g., through plan recognition.

[Poutakidis *et al.*, 2002] provides a method for tracking the progress of conversations using interaction protocols, and detection of some failures, using a Petri-net representation of the interaction protocols that are expected to take place

(rather than the expected failures as in the techniques discussed above). When protocols are matched against observations of messages, errors are detected. However, the representation has been shown to scale poorly with the number of agents [Gutnik and Kaminka, 2005].

[Kaminka and Tambe, 2000; Browning *et al.*, 2002] use a behavior-based approach. In a system consisting of n agents, each with m possible states, there exist $O(m^n)$ possible joint states. In this approach, the designer indicates the ideal state of coordination, by specifying the desired joint states, a subset of all possible joint states. The system observes the agents during run-time, and uses plan-recognition in order to infer their actual joint state. It then verifies that the actual joint state is indeed a desired one. [Kaminka and Bowling, 2002] present a scalable method for such assessment. Unfortunately their method only enables detection of system failures in cases where the desired joint states are in perfect agreement.

This paper presents a systematic approach to detecting coordination failures based on observation and plan-recognition. It utilizes a model-based approach, wherein the designer only specifies desired joint states, rather than all possible states of failure. The approach also takes accounts for the inherent uncertainty that exists when another's state is inferred, e.g., due to ambiguity in plan recognition. We show that we can compactly represent joint states using $n*m$ matrices, and thus reduce the potential $O(m^n)$ check to a $O(n \cdot m)$ check in many cases.

3 Motivation

As we mentioned above, the main weakness of earlier methods, was their exponential complexity [Browning *et al.*, 2002]. For example, consider a management system for a shop consisting of the following 6-agent crew: Annie the manager, Benny the cashier, two sellers — Canny and Danny, Ernie the storekeeper and a guard, Fanny. Agents may be in one of eight possible states: dealing with customers, handling equipment, taking a break etc. The agents may be placed in various pre-defined combinations, considering their states. For instance, the following combination is legitimate: {Annie: watch, Benny: sell, Canny: dealing, Danny: break, Ernie: equip, Fanny: guard}. However, the following combination is illegitimate: {Annie: watch, Benny: sell, Canny: negotiate, Danny: break, Ernie: sell, Fanny: guard}, since Ernie is forbidden to sell while he has no substitute. Thus, since any agent may be in any state, there are as many as 8^6 (262,144) possible combinations in this small shop. Any additional state or agent increases the complexity exponentially.

At run-time, the system is provided with a simple plan recognition capability that defines for each observed action, what states the agent may occupy. For example, if Danny is observed talking on the phone, he is either negotiation with a customer (state *negotiate*), or making a private call during his break time (state *break*). However, we are assured that he is not guarding at the moment. The system uses this mechanism to infer, for each agent, its possible states. The system then combines these individual states and indicates all possible joint states the agent may be in. These are then compared with the desired joint states list. The system can then determine whether a failure exists (i.e., none of the inferred possible states is a desirable state).

Nevertheless, the possibility of multiple joint states presents us with the difficulty of deciding whether a failure actually occurred since the list of inferred states contains desired and undesired states (i.e., ambiguous conclusions). This problem was addressed by [Kaminka and Tambe, 2000].

A key challenge involves the representation of potentially exponential inferred (and desired) joint states in a way that reduces matching run-time. Since the number of possible joint states in exponential, a simplistic approach would take exponential run-time.

Our research offers means of avoiding the problem of exponential number of combinations. This is done by encapsulating different combinations into simple, relatively small structures, called *extended-combinations,* or *e-combs.* For example, suppose that our coordination requires that when the guard, Fanny, is taking a break or talking to the manager, the storekeeper must replace her, and the sellers are allowed to deal with equipment at this situation. Suppose too, that similar logic is applied to the other agent, such that the following is legitimate: Fanny's state is break or inner-talk, Ernie's state is guard, Canny and Danny are either equipment, negotiate, sell or break, Benny's state is sell or break and Annie is either watching or inner-talking. The example defines 128 joint-state combinations. However, rather than enumerating them, we defined them implicitly in the description above. Using this kind of definition provide two significant features. First, the design and definition of desired coordination becomes much easier. Not only does it actually involve smaller structures, but it is also very logical and straight-forward. Second, and more importantly: the complexity no longer depends in the agents and states in an exponential manner.

4 A Matrix Representation Approach

In order to contextualize our solution, we will present the formalism of observation-based failure-detection [Browning *et al.*, 2002].

Let A be a set of n *agents*, $A = \{a_1, a_2, \ldots, a_n\}$, where each agent may be found at any time in one of m possible *states*, $S = \{s_1, s_2, \ldots, s_m\}$. At any given moment, the agents are at a given *joint state*, that is, each one of them is found in a specific state. Each joint state (which is also referred as a *combination*) may be represented as an n-tuple of states, $\langle s_{k_1}, s_{k_2}, \ldots, s_{k_n} \rangle$, $s_{k_i} \in S$, where s_{k_i} represents the state of agent a_i. A *system* is defined by agents, states, and the *allowed combinations*. These combinations, defined by the program designer, signify the states in which each agent is allowed to be.

Since the desired coordination is defined as a collection of combinations, the complexity of space needed is bound by the maximum number of combinations. Since each of the n agents may be in m possible states, the number of combinations is actually bounded by $O(m^n)$. Although this calculation represents a worst case scenario, the average case still increases exponentially as agents are added.

During run-time, the system is provided with the possible current states of each of the agents. This information is provided by an observer, who recognizes the agents particular

behavior in every state. Thus, one observation may be interpreted as many different states (and vice versa). The possible states selected by the agents, resembling the allowed combinations, are provided as a set of combinations.

Determining whether a failure has occurred is done by analyzing the desired coordination joint-states, and the hypothesized current states, and testing whether at least one joint state appears in both sets (this is an optimistic policy [Kaminka and Tambe, 2000]). Hence, the time complexity is equivalent to the space complexity. This is unacceptable when dealing with large systems. Using this approach, even a system consisting of a few dozens of agents, with a small number of states, will be difficult to monitor.

We present a new representation of the allowed combinations, called *extended combinations*, or *e-combs*. An e-comb is a Boolean matrix of order $n \times m$. Each row of the matrix is assigned to one agent and each column is assigned to one state. The allowed combinations are defined by the 1 elements in the matrix. That is, e-comb C defines all the combinations of the form:

$$a_1 : s_{k_1}, a_2 : s_{k_2}, \ldots, a_n : s_{k_n} \mid C_{i,k_i} = 1$$

Consider the example given in Section 3. Assume that the agents are numbered 1 to 6, according to alphabetical order (Annie is 1, Benny is 2 etc.). States are numbered 1 to 8 in the following order: break, idle, negotiate, sell, inner-talk, watch, guard, equipment. The appropriate ecomb would then be

$$D^{6\times 8} = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & [1] & 1 & 0 & 0 \\ [1] & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ [1] & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ [1] & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & [1] & 0 \\ [1] & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

This e-comb represents all the combinations in which agent a_i is found in some state s_j, such that $d_{i,j} = 1$. For example, a legal combination is $\{a_1 : s_5, a_2 : s_1, a_3 : s_1, a_4 : s_1, a_5 : s_7, a_8 : s_1\}$ (denoted by square brackets).

Let us now address the observations. In our system, aside for the logical states of agents, which are the states $s_k \in S$, there is also a definition of the various *observable actions* the agents may take, $B = \{b_1, b_2, \ldots, b_\ell\}$. Returning to the shop example once again, we will remember that while we had 8 states, we also had 9 actions ($\ell = 9$): talk (b_1), phone (b_2), stand (b_3), walk (b_4), counter (b_5), put (b_6), get (b_7), carry (b_8) and other (b_9). Every agent exists in one of these actions any given moment. However, this is not a one-to-one mapping. For example, if an agent is observed to be carrying a product (i.e., taking action b_8), it may do that either while making order in the shop (i.e., found in state s_8 – equipment), or it may carry it to a customer during a sell (s_4) state. In the opposite direction, during a sell (s_4) the agent may also take the action of sitting near the counter (b_5) or getting a product from the shelf (b_7).

We represent the relation between the agents state and its actions using a matrix, to abstract the plan recognition process. The matrix I (stands for *Interpretation*) is a Boolean matrix of order $m \times \ell$. In this matrix, the value of an element $I_{j,k}$ is '1' if once the agent's state is s_j it may take action b_k; in other words, if the agent is observed to be acting b_k, we may *interpret* it as being in (possibly) state s_j. Note that this says nothing about the plan recognition algorithm itself, other than that it requires it to be able to support the interpretation of what states are possible, given observed actions. In our example, this is how the interpretation matrix is defined for the shop system:

$$I^{8\times 9} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Row 4 of this matrix, for example, shows that the possible actions in state s_4 (sell) are b_5 (counter), b_7 (get) and b_8 (carry). Column 8 shows that if an agent is being observed as acting b_8 (carry), it may be in one of the states s_4 (sell) or s_8 (equipment).

During run-time, observation of agent action is conducted by an additional system with relevant sensors. At any given time, an agent is observed performing exactly one action. We represent it as a Boolean matrix W of order $n \times \ell$ (n represents the agents' size and ℓ the actions' size), called *observation matrix*. In this matrix, there is exactly one element which is '1' in each row. Thus, if $w_{i,k}$ is 1, it means that agent a_i is observed as acting b_k. For example, in the following observation matrix at time t,

$$W_t^{6\times 9} = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{array} \begin{array}{c} \begin{array}{ccccccccc} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 & b_8 & b_9 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Annie (a_1) was observed standing (b_3), as was Benny (a_2), Canny (a_3) is on the phone (b_2) etc.

The possible states in which each agent is found at that moment (t), then, are calculated using the formula:

$$\Omega_t = W_t \cdot I^T$$

where Ω_t is an $n \times m$ Boolean matrix (that is, an e-comb), in which each element j in row i represents whether it is possible that agent a_i is now in state s_j ('1' entry) or not ('0' entry). Note that each element $\Omega_{ti,j}$ is the sum of multiplying each element k in row i of W_t by element k in column j of I^T. This multiplication, of course, is '1' iff both of them are '1'. Since each row in W_t has exactly one element which is '1', the value of each element in Ω_t will be at most '1'.

In our previous example:

$$\Omega_t^{6\times 8} = W_t \cdot I^T = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

For example, our observation may lead us to conclude that Annie's state is either s_1 or s_2 or s_6 or s_7.

Using these matrices, we can now explain the failure detection algorithm. Failure is defined as a situation wherein none of an agents possible assigned state (according to Ω_t) appear on the 'allowed combination' list, designated as D

(the desired coordination e-comb). In order to examine possible matches we will operate a logical and between D and Ω_t in an element-by-element process, to get the results matrix, $R^{n\times m}$, $r_{i,j} = d_{i,j} \wedge \omega_{ti,j}$. Being a Boolean $n \times m$ matrix, R itself is in fact an e-comb.

R represents all the agents-assigned combinations that satisfy D according to observation. E-comb R represents all the combinations in which agent a_i is found in one of the states s_j that match '1' element in row R_i. Thus, if in each row i in R there is at least one '1' element, it implies that at least one combination exists. In this case, we may assume that the agents will be found in one of those joint states. If, However, R defines no combination, then the assigned agents states are definitely forbidden. In this case, a failure alert is warranted.

If at least one combination is found, R must include at least one assignment for each agent. In other words: at least one 1 element on each row. If an all-zero row exists, it indicates a no-assignment situation in R, in which case R does not define any combination. This operation takes only $O(nm)$ operations (counting the '1's for m elements on each of R's n rows). Returning to the shop example, matrix R will be the result of an element-by-element 'and' operation between the desired coordination D and the interpretation e-comb Ω_t, which were both presented earlier. This provides the matrix

$$R_t = \Omega_t \wedge D_t = \begin{array}{c} \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \left(\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right). \end{array}$$

in this e-comb, the two bottom lines, representing Ernie and Fanny, are all-zero. No desired combination can explain their actions. A failure has been detected.

5 Complex Coordination

One e-comb will usually not suffice for a full desired coordination definition. Thus, e-comb D that we introduced earlier, only partially defines the allowed combinations in the shop desired coordination. It deals only with Ernie replacing Fanny in guard duty.

However, we cannot add another state to D, by just changing $d_{6,7}$ (Fanny:guard) from 0 to 1. This would allow undesired combinations, such as Ernie and Fanny guarding simultaneously. Hence, we much provide a general notation that allows the definition of multiple types of coordination. The idea is to extend the e-comb approach so that it consists of more than one e-comb, without becoming exponentially complex.

5.1 E-combs Operators

The most important operator used to join a few e-combs is the 'or', which is notated as ' $\sqcup$ '. Defining two sets of coordination $D_1 \sqcup D_2$, means that the set of allowed combinations in the system is the union of all the combinations defined by D_1 and all the combinations defined by D_2. As long as Ω_t satisfies the 'none all-zero row' property with *either* D_1 or D_2 (or both, of course), there is no failure. This operator may be extended to longer expressions, of the kind $D_1 \sqcup D_2 \sqcup \cdots \sqcup D_p$.

We call this extended structure of combined e-combs using operators — a *rule*. Testing an interpretation e-comb Ω_t against a rule $\mathcal{R} = D_1 \sqcup D_2 \sqcup \cdots \sqcup D_p$ is simple. One must perform the 'all-zero' test presented earlier for each of the p e-combs. That is, for each D_k in $\mathcal{R}$, calculating the result matrix R_k by logically 'and'ing Ω_t with D_k in an element-by-element fashion, and then check whether R_k has an all-zero row or not. Due to the nature of the operator ' $\sqcup$ ', it is enough to verify that at least one such R_k has no all-zero row for assuming the agents are coordinated. Only if in each of the result matrices R_k there is an all-zero row, it indicates a coordination error; in this case, we are sure that the agents are not found in any allowed combination. Note that the complexity of such a simple rule, that involves no other operators than 'or', is $O(nmp)$, where p is the number of e-combs in the rule.

There may be cases in which use of $\sqcup$ will be less efficient, or more difficult for the designer. Thus, we present the second basic operator, 'and', which is notated by a ' $\sqcap$ '. The expression $D_1 \sqcap D_2$ represents all the combinations that are found in the intersection of those that are defined by D_1 and those defined by D_2. In other words, the 'non all-zero row' property for Ω_t must hold for *both* D_1 and D_2. In fact, one might notice that any expression of the form $D_1 \sqcap D_2$, may be reduced to an equivalent e-comb, that represents exactly the same set of combinations. This is the e-comb that is the result of a logical-and in an element-by-element fashion between D_1 and D_2.

In order to motivate the and operator, let us return to the shop example. Suppose that our shop, and an additional shop are running successfully and we would like the two shops to cooperate. The basic coordination rules of both shops are left untouched. However, now that two mangers are available, we add a constraint saying that one must always supervise the workers. At least one of the two managers must be watching (s_6) at any given time. Using previous methods, a new model would have required. E-combs with only an or operator might be easier, but will still require redesigning. This is due to the fact that the current system allows the manager to either watch or talk with its employees. Using the and operator, substantially simplifies our task.

Suppose that the shops use $\mathcal{R}_1$ and $\mathcal{R}_2$ as coordination rules. The first task would be to assemble all agents into one system. Instead of using e-combs of order 6×8 we use 12×8 e-combs, as described below. In order to avoid collision in agent names, we first renumber the agents of shop no. 2 as a_7 to a_{12}. We now have 12 agents,a_1 to a_{12}. Then, we must update definitions from both shops from a 6×8 domain to the new unified one. For this purpose, we expand each e-comb of $\mathcal{R}_1$ with six new rows, 7 to 18, which are all filled with 1s. This ensures that the desired coordination of the first shop is left untouched — since all rows, except for the first six, are defined as all ones. The other shop agents state is of no consequence, only the first 6 rows (agents) are meaningful. The same is done for the second shop rule, $\mathcal{R}_2$. In this case, we will expand the original e-combs in such a way that they will become rows 7 to 12 of the new e-combs, and fill rows 1–6 with all ones. Now, we have both shops running on the same system, each with its original rules. The only thing left

is to add the management restriction. This may be achieved by allowing one of following cases:

1. When manager 1, a_1, is watching the shop (s_6), the other manager, a_7, may either watch (s_6) or talk with its employees (s_5),
2. When manager 2, a_7, is watching the shop (s_6), the other manager, a_1, may either watch (s_6) or talk with its employees (s_5).

This is expressed by two e-combs; the first is

$$\mathcal{M}_1^{12\times 8} = \begin{array}{c} \\ a_1 \\ a_2 \\ \vdots \\ a_6 \\ a_7 \\ a_8 \\ \vdots \\ a_{12} \end{array} \begin{array}{c} \begin{array}{cccccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 & s_8 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array}.$$

We build $\mathcal{M}_2$ in a similar way. Then, we define the 'manager rule' to be $\mathcal{R}_M = \mathcal{M}_1 \sqcup \mathcal{M}_2$. Finally, we define the rule

$$\mathcal{R}_{\text{cooperative}} = \mathcal{R}_1 \sqcap \mathcal{R}_2 \sqcap \mathcal{R}_M.$$

The next section presents an algorithm for calculating of this kind of rules.

5.2 Computing Complex Rules

This section presents the general algorithm that tests a coordination rule which includes or and and operators against a given e-comb interpretation. The algorithm uses a *tree representation* of the rule. The leaves are the rules e-combs, and the inner nodes are the operators. The trees depth is then reduces until it consists of a simple or expression that can be easily calculated.

The first phase of the algorithm deals with the logical operators that construct the rule. The tree reduction is accomplished through *images*. An image represents, for each node in the tree, the possible combinations that are defined by the sub-tree whose this node is its root. The image is, in fact, one or more encapsulated e-combs. However, an image logically represents one node. In this way, we work our way up from the leaf nodes. The sub-tree of every node is replaced with an equivalent image.

The translation of a sub-tree is quite simple. It begins, recursively, from the root and runs in DFS until it reaches a leaf. On its way back, it replaces each node with an image. The manner, in which a node (sub-tree) is translated into an image, depends on the node-type. Since the sub-tree replacement is done during the DFS backtracking, the nodes offspring are already guarantied translation into images.

E-Combs: These are in fact the leaves of the tree; each e-comb node becomes an image which includes only one e-comb.

'Or' nodes: Each or node is replaced by an image that includes all the e-combs from the nodes image offspring.

'And' nodes: An and node that has a few image offspring performs according to the distribution law. It becomes an image that contains all the and combinations between e-combs from each of the offspring. In other words, if a node has k image offspring, each consisting of c_k different e-combs, then it will be replaced with an image that includes $\prod_{i=1}^{k} c_k$ e-combs. Each of those e-combs is built of a different combination of k e-combs, which are logically anded in an element-by-element fashion.

In order to demonstrate, let us refer to the following rule on some e-combs C_1 to C_9:

$$\mathcal{R} = C_1 \sqcup ((C_2 \sqcup C_3) \sqcap (C_4 \sqcup C_5)) \sqcup C_6 \sqcup (C_7 \sqcap C_8 \sqcap C_9)$$

Its tree form is represented in Fig. 1.

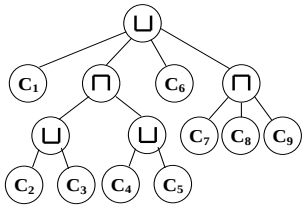

Figure 1: The Rule Tree for $\mathcal{R}$

The root has four offspring, two of which (the first and the third) are simple e-combs. The rightmost is an or node with three simple, e-combs offspring. The second one, is an and node, with two offspring, themselves sub-trees, each consisting of an or node and two e-combs offspring. We show how the algorithm reduces the tree, step by step. The first node is the leftmost node. It is, in fact, just a simple e-comb. It is therefore replaced by a simple image node that includes exactly this e-comb.

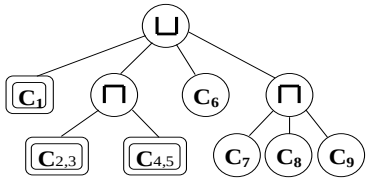

Figure 2: Rule Tree Reduction – step 1

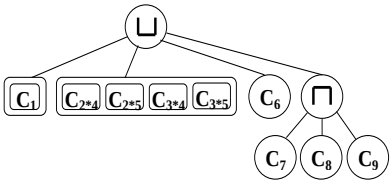

Figure 3: Rule Tree Reduction – step 2

In the next stage, the same thing is done to the next leaf (the e-comb C_2) and then to its sibling, C_3. Later, their parent node (of type 'or') becomes an image that includes both images (as a notational shortcut, we use the form $C_{2,3}$ as a substitution of C_2, C_3). The algorithm then continues the same

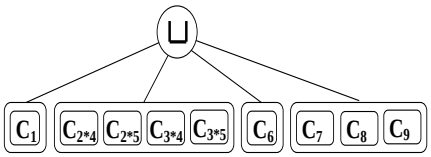

Figure 4: Rule Tree Reduction – step 3

process on the next sub-tree, and creates an image consisting of $C_{4,5}$ (Figure 2).

Next, we have an 'and' node, with two image offspring, each of which consists of two e-combs. As we saw earlier, the 'and' node is replaced by an image that includes all possible combinations of $\{C_2, C_3\}$ and $\{C_4, C_5\}$. These are the combinations $(C_2 \sqcap C_4), (C_2 \sqcap C_5), (C_3 \sqcap C_4), (C_3 \sqcap C_5)$, for short, $C_{2\star4}, C_{2\star5}, C_{3\star4}, C_{3\star5}$ (Figure 3). As was already mentioned, an e-combs 'and' ($\sqcap$) is in fact identical to an element-by-element 'and'. Hence, each of the expressions $C_{x\star y}$ is in fact one e-comb. During the next stage, the node of C_6 is replaced by an image with only this e-comb. Then the rightmost 'or' node, with three offspring (C_7, C_8, C_9) is replaced with one image of those three e-combs (Figure 4). At this stage, we reach the root 'or' node, which has four image offspring.

After reducing the whole tree, we are left with one image. This image includes multiple e-combs. Thus, in fact, it may be treated as a collection of e-combs that are all combined by an 'or' (' $\sqcup$ ') operator. As we noted earlier, a failure is detected if for all of them, the result of 'and'ing with Ω_t provides an e-comb with an all-zero row.

5.3 Complexity

As we saw in section 4, the complexity of a rule that consists of or operators only is $O(mnp)$, where p is the number of e-combs in the rule. The complexity of a rule that combines or and and cannot be described by a simple formula, and is highly related to the structure of the rule. However, generally, it grows linearly in the number of agents (n) and the number of possible states (m). The main factor is the complexity of the rule tree. Of course, common sense dictates that the more agents and states get involved, the greater the complexity of the rule. However, it does not necessarily grow exponentially. This is a very important property, since the complexity of other approaches is exponential in the number of agents, regardless the structure of the allowed combinations.

6 Summary and Future Work

In this paper we presented a new formal approach to observation-based fault detection. We defined a new matrix-based notation—the e-combs—which serves as a general framework for coordination design and definition in multi agent systems. At run-time the observer of the multi-agent system builds a similar matrix of the hypothesized states selected by the agents. Using this representation, we showed an efficient fault detection algorithm in a space and time complexity that is linear by the number of agents and state. The space and time needed for this algorithm are mainly dependent in the complexity of the rule—how many e-combs involve and in what kind of relations.

This research is novel in that it presents a solution that is both general and efficient for large-scale teams. It also eases the design of coordination requirements and allows modularity and reuse of already existing systems.

In the future we plan to add partial observations capabilities which will find the minimum set of agents that will together provide the complete information, or at least the best possible information. Combining this with explicit communication among agents may result a system that is cheap in resources, yet very reliable. In addition, at the moment our algorithm assumes that the coordination among the team members is defined at the beginning and must be consistent along the system lifetime. However, real-world multi-agent systems are dynamic, and the desired coordination may change, so we plan to extend our algorithm to dynamic coordination.

References

[Browning *et al.*, 2002] Brett Browning, Gal Kaminka, and Manuela Veloso. Principled monitoring of distributed agents for detection of coordination failures. In *Proceedings of Distributed Autonomous Robotic Systems 6*, pages 319–328. Springer-Verlag, 2002.

[Gutnik and Kaminka, 2005] G. Gutnik and G. A. Kaminka. A scalable petri-net representation of interaction protocols for overheaing. In *Developments in Agent CommunicationLNAI Volume 3396, van Eijk, R.; Huget, M. P. and Dignum, F. (Eds), Springer-Verlag. In press*, 2005.

[Horling and Lesser, 1999] Bryan Horling and Victor Lesser. Using Diagnosis to Learn Contextual Coordination Rules. *Proceedings of the AAAI-99 Workshop on Reasoning in Context for AI Applications*, pages 70–74, July 1999.

[Horling *et al.*, 2001] Bryan Horling, Brett Benyo, and Victor Lesser. Using Self-Diagnosis to Adapt Organizational Structures. *Proceedings of the 5th International Conference on Autonomous Agents*, pages 529–536, June 2001.

[Kaminka and Bowling, 2002] Gal A. Kaminka and Michael Bowling. Towards robust teams with many agents. *in Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-02)*, 2002.

[Kaminka and Tambe, 2000] Gal A. Kaminka and Milind Tambe. Robust multi-agent teams via socially-attentive monitoring. *Jornal of Artificial Intelligence Research*, 12:105–147, 2000.

[Klein and Dellarocas, 1999] Mark Klein and Chris Dellarocas. Exception handling in agent systems. *Proceeding of the Third International Conference on Autonomous Agents*, May 1999.

[Kraus *et al.*, 1998] Sarit Kraus, Sycara Katia, and Amir Evenchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104(1–2):1–69, 1998.

[Parker, 1998] Lynne E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[Poutakidis *et al.*, 2002] D. Poutakidis, L. Padgham, and M. Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. *in Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-02)*, 2002.