# Matrix-Based Representation for Coordination Fault Detection: A Formal Approach

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

## ABSTRACT

Teamwork requires that team members coordinate their actions. The representation of the coordination is a key requirement since it influences the complexity and flexibility of reasoning team–members. One aspect of this requirement is detecting coordination faults as a result of intermittent failures of sensors, communication failures, etc. Detection of such failures, based on observations of the behavior of agents, is of prime importance. Though different solutions have been presented thus far, none has presented a comprehensive and formal resolution to this problem. This paper presents a formal approach to representing multi-agent coordination, and multi-agent observations, using matrix structures. This representation facilitates easy representation of coordination requirements, modularity, flexibility and reuse of existing systems. Based on this representation we present a novel solution for fault-detection that is both generic and efficient for large-scale teams.

## 1. INTRODUCTION

Autonomous agents within multi-agent systems interact and coordinate to achieve their goals [4, 2]. The representation of the coordination is a key requirement since it may influence the complexity, modularity and the flexibility of reasoning operations of the team [10, 6].

One aspect of this requirement is detecting coordination faults. The increased deployment of robotic and agent teams in complex dynamic settings, has in turn, led to an increasing need for coordination failure handling [9, 8, 3, 5]. Coordination-failure detection does not indicate whether the group is achieving its goals but only if agent-coordination exists. Detection of coordination failures is essential for a recovery process during which cooperation is reinstated (e.g., a negotiations).

The ability to detect failures is affected by the representation of the coordination. A näive approach, for example, which maintains and reasons about the whole coordination space is is time consuming and takes up a lot of space. The motivation for this paper is to find a simple, efficient and modular representation which will enable to represent the coordination easily.

*This research was supported in part by BSF grant #2002401.


*AAMAS'07,* May 14–18, 2007, Honolulu, Hawai'i, USA.


Previous coordination-failures detection methods have adopted an approach using a causal-model which describes a pre-defined set of faults, i.e. a set of forbidden coordination relations [8, 3]. Obviously, this approach is not complete since it does not guarantee that all the possible failures are pre-defined. Others are only able to capture specific coordination failures, such as disagreements [7, 6]. The drawbacks of these approaches are caused by the representation model of the coordination. None of the works have taken a systematic approach to addressing this challenge. We, on the other hand, adopt a model-based approach by modeling the normal behavior of a team, i.e. the permitted coordination among the agents.

We present an efficient matrix-based approach to represent: (1) pre-defined coordination and (2) the agents' states as inferred from their observed actions. This representation has several benefits. First, it provides an easy and intuitive way to define the coordination between teammates. Second, unlike hierarchical structures, which strictly define relations in the organization, our approach allows flexible and complex relations. For example, under certain circumstances, it enables the inclusion of an agent in two subgroups. Third, since we do not represent the relations between teammates explicitly, but gather them compactly (joint coordination in the same matrix structure), this approach is scalable in the number of agents. Finally, the use of a matrix-based representation, enables the use of the matrix operations and yields interesting information about the agents. To summarize, the matrix representation enables an easy and efficient way to implement functions on team, like plan recognition, failure detection, diagnosis and other reasoning operations.

To demonstrate the new representation, we will present a failure-detection algorithm based on the matrix representation. The algorithm (described in Section 4) uses matrix operations to efficiently find coordination failures. 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. In section 3, we present our new notation to the coordination-definition and observation modeling, based on matrices. Section 4 uses this notation to present fault detection algorithm. An extension of the coordination model for complex systems is presented in section 5. An implementation of it in hierarchy models is presented in Section 6. Section 7 extends matrix-based representation for dynamic systems. At last, section 8 summarizes.

## 2. RELATED WORK

Horling et al. [3] present 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



978-81-904262-7-5 (RPS) 

enables it to detect and diagnose failures.

A similar approach has been presented by Klein and Dellarocas [8] 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. In this method, as the former, the failure-model approach dictates that all possible failures be analyzed in advance.

Kaminka at al. [7, 1] use a behavior-based approach using a hierarchical model. 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. 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 [6] present a scalable method for such assessment. Unfortunately, their method only enables detection of system failures in cases where the desired joint states are those of perfect agreement. In addition, their hierarchical representation of the coordination restricted the organizational relations between the agents.

This paper presents a formalization to represent team coordination based on matrix structures. This representation enables a systematic approach to detecting coordination failures based on observation and plan-recognition. It utilizes a model-based approach, wherein the designer specifies only desired joint states, rather than an abductive approach which defines all possible states of failure. Our approach also addresses uncertainty that exists due to ambiguity in plan recognition. We show that we can compactly represent joint states using $O(nm)$ matrix order, and thus reduce the potential $O(m^n)$ check to a $O(nm)$ check in many cases.

# 3. FUNDAMENTAL OBJECTS

This section presents the basic objects used in multi-agent systems and the relations among them. The most fundamental entities are the *agents*. At any moment, each agent is found in a given *state*. This is a logical, internal representation of the agent status, or belief, at this very moment. Throughout the next sections, we will refer to the following sets:

(i) Let $A$ be a set of $n$ agents, $\{a_1, a_2, ..., a_n\}$.

(ii) Let $S$ be set of $m$ states, $\{s_1, s_2, ..., s_m\}$.

For example, consider a management system for a shop consisting of the following six agents (hereinafter this example will be referred as "the shop "): ANNY the manager, BENNY the cashier, two sellers — CANNY and DANNY, ERNY the storekeeper and a guard, FRENNY:

$A_{\text{shop}}$={ANNY, BENNY, CANNY, DANNY, ERNY, FRENNY}

Agents may be in one of eight possible states:

$S_{\text{shop}}$={BREAK, IDLE, NEGOTIATE, SELL, INNERTALK, WATCH, GUARD, EQUIP}

Having the two sets $A$ and $S$, we can define the environment for a team:

**Definition 1 (environment).** Let $A$ be a set of agents, and let $S$ be a set of states. The pair $E = \langle A, S\rangle$ is called the *environment* of $A$ over $S$.

**Definition 2 (environmental-space).** Let $E = \langle A, S\rangle$ be an environment. The Cartesian product $A \times S$ is called the *environmental space* of $E$, and is denoted $E^+$.

The environmental space is, in fact, the set of all the possible $\langle$agent, state$\rangle$ pairs. Any of these pairs may define a single state in which an agent is found, as we will describe later. In addition to pairing each agent to one state, we would like to define a space which allows each agent to be paired with as many states as needed. For that purpose, we define the following space:

**Definition 3 (environmental-power-space).** Let $E = \langle A,\ S\rangle$ be an environment. The Cartesian product $A \times (\|S\|)$ (i.e., $A$ multiplied by all the subsets of $S$) is called the *environmental-power-space* of $E$, and is denoted $E^\times$.

Now that we have the definition of the environment and its associated spaces, we can continue with structures on those spaces. We refer an agent by its state:

**Definition 4 (position).** Let $E = \langle A, S\rangle$ be an environment. A pair $\langle a', s'\rangle \in E^+$ is called a *position* over $E$.

We could also attribute multiple states to one agent. For example, in case we are not sure what is the current state of the agent we will refer that agent superposition:

**Definition 5 (superposition).** Let $E = \langle A, S\rangle$ be an environment. A pair $\langle a', S'\rangle \in E^\times$ (i.e., $a' \in A$ and $S' \subseteq S$) is called a *superposition* over $E$.

For example, let us refer back to the agents and states presented in the shop . The pair $\langle$ERNY, GUARD$\rangle$ is a position, while $\langle$ANNY, {INNERTALK, WATCH}$\rangle$ is a superposition.

Having each agent found in a particular state defines a unique coordination among the agents. In order to save the generic nature of it, we define the coordination as a function.

**Definition 6 (coordination).** A *coordination* function over an environment $\langle A, S\rangle$ is a function that *positions* each agent in a particular state: $\gamma : A \rightarrow S$

For example, in the shop example above, an allowed coordination is {$\langle$ANNY, WATCH$\rangle$, $\langle$BENNY, SELL$\rangle$, $\langle$CANNY, NEGOTIATE$\rangle$, $\langle$DANNY, BREAK$\rangle$, $\langle$ERNY, GUARD$\rangle$, $\langle$FRENNY, INNERTALK$\rangle$}. However, the following coordination is illegitimate: {$\langle$ANNY, WATCH$\rangle$, $\langle$BENNY, SELL$\rangle$, $\langle$CANNY, NEGOTIATE$\rangle$, $\langle$DANNY, BREAK$\rangle$, $\langle$ERNY, SELL$\rangle$, $\langle$FRENNY, GUARD$\rangle$}.

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of agents and $S = \{s_1, s_2, \ldots, s_m\}$ be a set of states. We can represent the position of the agents by a Boolean matrix of order $n \times m$. This matrix represents an extended combination of the agents' positions (e-comb, for short):

**Definition 7 (extended-combination).** Let $E$ be the environment $\langle A,\ S\rangle$. An *extended-combination* (or *e-comb* for short) $C$ over $E$ is a Boolean matrix of order $n \times m$ ($C \in \mathbb{B}^{n\times m}$) provides:

$$c_{ij} = \begin{cases} 1 & \gamma(a_i) = s_j \\ 0 & \text{otherwise} \end{cases}$$

While coordination positions each agent in a particular state, we sometimes need to position each agent in one of a few possible states (based on Definition 5).

**Definition 8 (supercoordination).** A *supercoordination* function over some environment $E = \langle A,\ S\rangle$ is a function $\Gamma : A \rightarrow \|S\|$ i.e., it positions each agent in a *set* of possible states. Representing it by e-comb provides:

$$c_{ij} = \begin{cases} 1 & s_j \in \Gamma(a_i) \\ 0 & \text{otherwise} \end{cases}$$

Figure 1 presents an example for such a function, and Figure 2 presents its appropriate e-comb. The rows represent the agents and the columns represent the states. This representation allows defining multiple constraints between the agents in the same structure.

$$\Gamma(a) = \begin{cases} \{\text{InnerTalk, Watch}\} & a = \text{Anny} \\ \{\text{Break, Sell}\} & a = \text{Benny} \\ \left\{\begin{matrix}\text{Break, Negotiate,} \\ \text{Sell, Equip}\end{matrix}\right\} & a \in \{\text{Canny, Danny}\} \\ \{\text{Guard}\} & a = \text{Erny} \\ \{\text{Break, InnerTalk}\} & a = \text{Frenny} \end{cases}$$

**Figure 1: A supercoordination function.**

$$C^{6\times 8} = \begin{array}{c} \\ \text{Anny} \\ \text{Benny} \\ \text{Canny} \\ \text{Danny} \\ \text{Erny} \\ \text{Frenny} \end{array} \begin{array}{c} \begin{array}{cccccccc} \text{Break} & \text{Idle} & \text{Negotiate} & \text{Sell} & \text{InnerTalk} & \text{Watch} & \text{Guard} & \text{Equip} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

**Figure 2: The e-comb representation of the supercoordination of Figure 1.**

For example, one coordination constraint could be ⟨Anny selects state InnerTalk while Benny selects Sell⟩. Another coordination that is represented by this e-comb is ⟨Anny could select Watch, while concurrently, Benny selects state Sell⟩.

A special kind of supercoordination is one that does not assign any state to at least one of the agents. In other words, a supercoordination which assigns at least one agent the empty-set. We call this an ill supercoordination. In the e-comb it will be represented by a row of zeros.

**Definition 9 (ill–supercoordination).** Let $\Gamma$ be a supercoordination function over some environment $E = \langle A,\ S\rangle$. Then $\Gamma$ is an *ill–supercoordination* iff $\exists\ a\ \in\ A\ \mid\ \Gamma(a)\ =\ \emptyset$. Using the e-comb representation: $\exists i,\ 1 \leq i \leq n:\ \bigvee_{j=1}^{m} c_{ij} = 0$

The example shown in Figure 1, for instance, is not an ill–supercoordination, since it positions all agents to non–zero rows (Figure 2).

At any given moment, each agent is in a given *state*. As a result of its state, each agent takes some *action*, in order to fulfill its goal. An action is visible, i.e. others might observe it. A state is not necessarily related to one particular action. Rather, it is possible that one of a few given actions will be taken at service of the same state. In the opposite direction, the same action might be taken at service of a few different states. We will annotate the actions as a set $B = \{b_1, b_2, \ldots, b_\ell\}$.

For example, in the shop we define eight states logical positions of the agents and nine actions, which the agents might act upon. State Sell, for example, is when an agent is busy with closing the deal with a customer. Positioned at this state, the agent might act in one of the actions Get (getting the product off the shelf), Carry (carrying it to the customer) or Counter (sitting near the counter). On the other hand, an agent might also Carry or Get while positioned at state Equip, and not only when positioned in Sell.

When designing a multi-agent system, the designer defines which actions might be taken by an agent when positioned in each state. This is called the *latitude* of the agent.

**Definition 10 (latitude).** Let $E = \langle A, S\rangle$ be an environment, and $B$ be a set of actions, the *latitude* of agent $a \in A$ is a function $\lambda_a : S \to \|B\|$.

This function maps, for a given agent $a \in A$, each state to a subset of actions which the agent is allowed to pick while being in this state. The straight-forward inverse function of $\lambda_a$, the function $\lambda_a^{-1}$, would map subsets of $B$ to elements in $S$. While this function is not interesting, we do define a kind of 'inverse' to the latitude function:

$$\lambda(s) = \begin{cases} \{\text{Talk, Phone, Stand, Other}\} & \text{Break} \\ \{\text{Stand}\} & \text{Idle} \\ \{\text{Talk, Phone}\} & \text{Negotiate} \\ \{\text{Get, Carry, Counter}\} & \text{Sell} \\ \{\text{Talk}\} & \text{InnerTalk} \\ \{\text{Stand, Walk, Talk}\} & \text{Watch} \\ \{\text{Stand, Walk}\} & \text{Guard} \\ \{\text{Walk, Carry, Put, Get}\} & \text{Equip} \end{cases}$$

(a) A latitude function

$$\Lambda(b) = \begin{cases} \left\{\begin{matrix}\text{Break, Negotiate,} \\ \text{InnerTalk, Watch}\end{matrix}\right\} & \text{Talk} \\ \{\text{Break, Negotiate}\} & \text{Phone} \\ \left\{\begin{matrix}\text{Break, Idle,} \\ \text{Watch, Guard}\end{matrix}\right\} & \text{Stand} \\ \{\text{Watch, Guard, Equip}\} & \text{Walk} \\ \{\text{Sell}\} & \text{Counter} \\ \{\text{Equip}\} & \text{Put} \\ \{\text{Sell, Equip}\} & \text{Get} \\ \{\text{Sell, Equip}\} & \text{Carry} \\ \{\text{Break}\} & \text{Other} \end{cases}$$

(b) An interpretation function

**Figure 3: A latitude function for the example of the shop , and its interpretation function.**

$$I^{9\times 8} = \begin{array}{c} \\ \text{Talk} \\ \text{Phone} \\ \text{Stand} \\ \text{Walk} \\ \text{Counter} \\ \text{Put} \\ \text{Get} \\ \text{Carry} \\ \text{Other} \end{array} \begin{array}{c} \begin{array}{cccccccc} \text{Break} & \text{Idle} & \text{Negotiate} & \text{Sell} & \text{InnerTalk} & \text{Watch} & \text{Guard} & \text{Equip} \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

**Figure 4: The interpretation-matrix for the interpretation function presented in Figure 3.**

**Definition 11 (interpretation).** Let $E = \langle A, S\rangle$ be an environment, and $B$ be a set of actions, the *interpretation* of agent $a \in A$ is the function $\Lambda_a : B \to \|S\|$.

$\Lambda_a$ of a given action $b$, is the set of all states that have $b$ in their latitude. Given an action of an agent $a'$, we *interpret* its action as one of a few given states, using this function. Figure 3 presents the latitude and interpretation functions for the shop example.

Given a set of states $S = \{s_1, s_2, \ldots, s_m\}$ and a set of actions $B = \{b_1, b_2, \ldots, b_\ell\}$, we can represent the interpretation of the actions to the states by a Boolean matrix of order $\ell \times m$.

**Definition 12 (interpretation-matrix).** Let $S$ be a set of states and $B$ a set of actions, an *interpretation-matrix* $I$ from $B$ to $S$ is a Boolean matrix of order $\ell \times m$ ($I \in \mathbb{B}^{\ell\times m}$) provides:

$$i_{ij} = \begin{cases} 1 & s_j \in \Lambda(b_i) \\ 0 & \text{otherwise} \end{cases}$$

Figure 4 presents the appropriate interpretation-matrix to the interpretation function presented in Figure 3. The rows represent the actions and the columns represent the states. For example, the second row says that once an agent is observed doing Phone, then its state is one of {Break, Negotiate}.

Knowing the exact state of each agent at every time requires that the agent reports its state any time it is changed. This is usually infeasible, since it involves massive communication resources. Our

$$\Theta^{6\times 9} = \begin{array}{c} \\ \text{ANNY} \\ \text{BENNY} \\ \text{CANNY} \\ \text{DANNY} \\ \text{ERNY} \\ \text{FRENNY} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \text{TALK} & \text{PHONE} & \text{STAND} & \text{WALK} & \text{COUNTER} & \text{PUT} & \text{GET} & \text{CARRY} & \text{OTHER} \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

**Figure 5: An observation matrix.**

$$\Omega^{6\times 8} = \Theta \cdot I = \begin{array}{c} \\ \text{ANNY} \\ \text{BENNY} \\ \text{CANNY} \\ \text{DANNY} \\ \text{ERNY} \\ \text{FRENNY} \end{array} \begin{array}{c} \begin{array}{cccccccc} \text{BREAK} & \text{IDLE} & \text{NEGOTIATE} & \text{SELL} & \text{INNERTALK} & \text{WATCH} & \text{GUARD} & \text{EQUIP} \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

**Figure 6: The e-comb given by the product between the observation matrix and the interpretation matrix.**

observation-based approach suggests looking at the action of each agent. Thus the last building block we define is the observation.

**Definition 13 (observation).** Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of agents and $B = \{b_1, b_2, \ldots, b_\ell\}$ a set of actions, an *observation* is a function $\omega : A \rightarrow B$, that maps each agent to a particular action. $\Theta$ stands for the observation matrix representation:

$$\theta_{ij} = \begin{cases} 1 & \omega(a_i) = b_j \\ 0 & \text{otherwise} \end{cases}$$

Figure 5 presents an example to an observation matrix. The rows represent the agents and the columns the actions. Pay attention that in every row there is exactly a single '1' since every agent is observed in one action.

# 4. A CASE STUDY — FAULT DETECTION

In the former section we defined a formalism to a matrix–based representation for team coordination. We defined an extended coordination matrix between the agents' states (e-comb—Definition 7), an interpretation matrix for inferring the current states of the agents by their actions (Definition 12) and an observation matrix which maps between the agents and their current observed actions (Definition 13). In this Section we will present a case study, fault detection, in order to show the benefits of such representation.

A coordination fault occurs when the current agents' positions (Definition 4) do not match the expected coordination given by the e-comb (Definition 7). Thus, if we know the current positions of the agents, we can say for sure whether the system has a fault or not. The exact state of each agent is known only to the agent itself. However, its action is observable. By observing its current action, we can infer the state in which the agent is found. This could be done using the formula: $\Omega = \Theta \cdot I$ ($\Theta$ is the observation matrix and $I$ is the interpretation matrix), where $\Omega$ is an $n \times m$ Boolean matrix (that is, an e-comb). 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 $\omega_{i,j}$ is the sum of multiplying each element $k$ in row $i$ of $\Theta$ by element $k$ in column $j$ of $I$. This multiplication, of course, is '1' iff both of them are '1'. Since each row in $\Theta$ has exactly one element which is '1', the value of each element in $\Omega$ will be at most '1'.

For example, figure 6 presents the e-comb given by the product between the observation matrix (given in Figure 5) and the interpretation matrix (given in Figure 4). Our observation may lead us to conclude that CANNY's state is either BREAK or NEGOTIATE.

$$R = \Omega \wedge C = \begin{array}{c} \\ \text{ANNY} \\ \text{BENNY} \\ \text{CANNY} \\ \text{DANNY} \\ \text{ERNY} \\ \text{FRENNY} \end{array} \begin{array}{c} \begin{array}{cccccccc} \text{BREAK} & \text{IDLE} & \text{NEGOTIATE} & \text{SELL} & \text{INNERTALK} & \text{WATCH} & \text{GUARD} & \text{EQUIP} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

**Figure 7: The e-comb given by boolean 'and' operation between the desired coordination $C$ and the interpretation e-comb $\Omega$.**

We can now explain the failure detection algorithm. Failure is defined as a situation wherein none of an agent's possible assigned state (according to $\Omega$) appears on the 'allowed coordination', designated as $C$ (the desired coordination e-comb). In order to examine possible matches we will operate a logical 'and' between $C$ and $\Omega$ in an element-by-element process, to get the results matrix, $R^{n\times m}$, $r_{i,j} = c_{i,j} \wedge \omega_{i,j}$. Being a Boolean $n \times m$ matrix, $R$ itself is in fact an e-comb.

$R$ represents all the agents-assigned combinations that satisfy $C$ according to interpreted states by the observation. The combinations represented by $R$ are all those that 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 ill–supercoordination (Definition 9), meaning, an all-zero row exists, then the assigned agents' states are definitely forbidden. In this case, a failure alert is warranted. 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 $R$ in Figure 7 is the result of an element-by-element 'and' operation between $C$ (Figure 2) and $\Omega$ (Figure 6). In this e-comb, the two bottom lines, representing ERNY and FRENNY, are all-zero. No desired combination can explain their actions. A failure has been detected.

In order to detect failures by observations only, we define two policies of decision [7]. The *optimistic policy* assumes that as long as the system is not proven to be faulty, no fault should be reported. Using this policy, one can never get a false alarm. If it reports a fault, then a fault has certainly occurred. The other policy is the *pessimistic policy*. This policy reports a fault in the system, unless it is completely confident that no fault has occurred. Using this policy, one can never get to a situation of an unreported fault. We have adopted here an optimistic policy, thus in matrix $\Omega$ we inferred *all* the possibilities of the states that could be taken by the observed agents. By generating the result matrix ($R$) we check if at least one of the interpreted joint–states of the observed agents is consistent with the desired coordination.

Sometimes, an agent cannot detect the exact action of one of its teammates. In this case, we can still provide a partial solution; the agent may assume that the teammate row is 'all-ones' (i.e., its action might be any action in the system). Although in this case we are likely to miss faults, we still keep the property of the optimistic policy, that is, report no false-alarms. If the system principally allows communication between agents, the agent may better solve the problem by explicitly communicate agents whose action are not observable for it.

# 5. COMPLEX COORDINATION

One e-comb will usually not suffice for a full desired coordination definition. Thus, the e-comb we introduced earlier (Definition 7), may only partially define the allowed combinations in a desired

coordination. For instance, in the the shop example, assume ERNY could replace FRENNY in GUARD duty, the e-comb in Figure 2 does not deal with this new relation. Moreover, we cannot add another state to $C$, by just changing $c_{6,7}$ $\langle$FRENNY, GUARD$\rangle$ from '0' to '1'. This would allow undesired combinations, such as ERNY and FRENNY guarding simultaneously. Hence, we must provide a general notation that allows the definition of multiple types of coordination. The idea is to extend the e-comb 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', notated as ' $\sqcup$ '. Defining two sets of coordination $C_1 \sqcup C_2$, means that the set of allowed combinations in the system is the union of all the combinations defined by $C_1$ and all the combinations defined by $C_2$. As long as $\Omega$ satisfies the property of none ill–supercoordination (Definition 9) with *either* $C_1$ or $C_2$ (or both, of course), there is no failure. This operator may be extended to expressions of the kind $C_1 \sqcup C_2 \sqcup \cdots \sqcup C_p$.

We call this extended structure of combined e-combs using operators a *rule*. Testing an interpretation e-comb $\Omega$ against a rule $\mathcal{R} = C_1 \sqcup C_2 \sqcup \cdots \sqcup C_p$ is simple. One must perform the ill–supercoordination test presented earlier for each of the $p$ e-combs. That is, for each $C_k$ in $\mathcal{R}$, calculating the result matrix $R_k$ by logically 'and'ing $\Omega$ with $C_k$ in an element-by-element fashion, and then check whether $R_k$ is an ill–supercoordination e-comb. Due to the nature of the operator ' $\sqcup$ ', it is enough to verify that at least one such $R_k$ is not an ill–supercoordination e-comb, in order to conclude that the agents are coordinated. 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 the use of $\sqcup$ is less efficient, or more difficult for the designer. Thus, we present the second basic operator, 'and', which is notated by a ' $\sqcap$ '. The expression $C_1 \sqcap C_2$ represents all the combinations that are found in the intersection of those that are defined by $C_1$ and those defined by $C_2$. In other words, the absence of the property of ill–supercoordination for $\Omega$ must hold for *both* $C_1$ and $C_2$. In fact, one might notice that any expression of the form $C_1 \sqcap C_2$, 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 $C_1$ and $C_2$. The complexity of computing an 'and' rule is also $O(nmp)$, where $p$ is the number of e-combs in the rule.

Let us motivate the 'and' operator by an example. Suppose that our shop, and an additional shop with a set of six agents $A_2 = \{a_7, a_8, a_9, a_{10}, a_{11}, a_{12}\}$ and the same set of states as in our 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 managers are available, we add a constraint saying that one must always supervise the workers, i.e. at least one of the two managers must be watching (WATCH) at any given time. Using previous methods, a new model would have been required. E-combs with only 'or' operators 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 to 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, correspondingly. The first task would be to assemble all agents into one system. Since we joint the agents in the two shops to a one big shop, there are now 12 agents. Instead of using e-combs of order $6 \times 8$ we use $12 \times 8$ e-combs. Then, we must update definitions from both shops from a $6 \times 8$ domain to the new unified one. For this purpose, we expand each e-comb of $\mathcal{R}_1$ with six new rows, 7 to 12, which are all filled with '1's. 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 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 the following cases:

1. When manager 1, ANNY, is watching the shop (WATCH), the other manager, $a_7$, may either watch (WATCH) or talk with its employees (INNERTALK),
2. When manager 2, $a_7$, is watching the shop (WATCH), the other manager, ANNY, may either watch (WATCH) or talk with its employees (INNERTALK).

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

$$\mathcal{M}_1^{12\times 8} = \begin{array}{c} \\ \text{ANNY} \\ \text{BENNY} \\ \vdots \\ \text{FRENNY} \\ a_7 \\ a_8 \\ \vdots \\ a_{12} \end{array} \begin{array}{c} \begin{array}{cccccccc} \text{BREAK} & \text{IDLE} & \text{NEGOTIATE} & \text{SELL} & \text{INNERTALK} & \text{WATCH} & \text{GUARD} & \text{EQUIP} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array}.$$

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 which merges the rule of our shop ($\mathcal{R}_1$) with the rule of the new shop ($\mathcal{R}_2$) and the joint rule of the managers ($\mathcal{R}_M$): $\mathcal{R}_{\text{cooperative}} = \mathcal{R}_1 \sqcap \mathcal{R}_2 \sqcap \mathcal{R}_M$.

The next section presents an algorithm for calculating 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 rule's e-combs, and the inner nodes are the operators. The algorithm traverses the tree in bottom–up and unifies e-combs, reducing its depth. The tree's depth is incrementally reduced 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 follows depth–first 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 depth–first backtracking, the node's offsprings are already guarantied translation into images. Let us introduce the types of image:

**[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 node's 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 $k$ images 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 'and'ed 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. 8. The root has four offsprings, two of which (the first and the third) are simple e-combs. The rightmost is an 'and' node with three simple, e-combs offsprings. The second one, is an 'and' node, with two offsprings, themselves subtrees, each consisting of an 'or' node and two e-combs offspring.

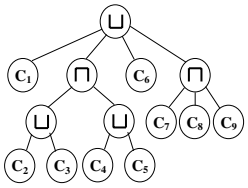


**Figure 8: The Rule Tree for $\mathcal{R}$.**

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.

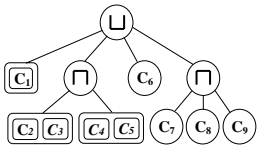


**Figure 9: Rule Tree Reduction – step 1.**

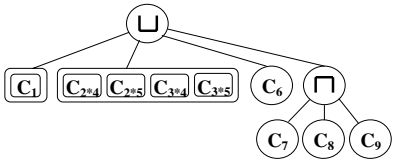


**Figure 10: Rule Tree Reduction – step 2.**

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. The algorithm then continues the same process on the next sub-tree, and creates an image consisting of $(C_4, C_5)$ (Figure 9).

Next, we have an 'and' node, with two images offspring, each of which consists of two e-combs. As we have seen 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 10). As we have already mentioned, 'and'ing e-combs ( $\sqcap$ ) is in fact identical to an element-by-element 'and'. Hence, each of the expressions $C_{x\star y}$ is a one

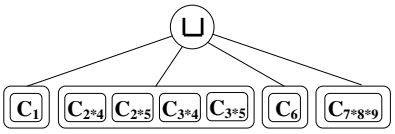


**Figure 11: Rule Tree Reduction – step 3.**

e-comb. During the next stage, the node of $C_6$ is replaced by an image with only this e-comb. Then the rightmost 'and' node, with three offsprings $(C_7, C_8, C_9)$ is replaced with an image of one e-comb, which is the result of 'and'ing those three e-combs — $C_{7\star8\star9}$ (Figure 11). At this stage, we reach the root 'or' node, which has four images offsprings.

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 $\Omega$ provides an e-comb with an all-zero row.

The complexity of rule-tree reducing cannot be described by a simple formula, and is highly related to the structure of the rule (it is out of the scope of this paper). However, this process is done offline, once, after the rule is defined. Therefore, it does not affect the complexity of the run-time fault detection algorithm itself. This complexity is left $O(nmp)$, where $n$ is the number of agents, $m$ is the number of states and $p$ is the number of e-combs in the reduced tree image. The important property of this complexity is, that for a given form of rule, $p$ is fixed. Therefore, for a given structure of rule, the complexity grows linearly in the number of agents and states in the system — unlike other approaches, which are exponential in the number of agents and states.

# 6. HIERARCHICAL STRUCTURES

Following the previous section, one of the benefits of using e-combs is that defining a desired complex coordination becomes easier. In order to demonstrate that on at least one wide-used system structure we will show, in this section, how to represent a hierarchical structure [10] using the e-combs concept.

We see a few drawbacks in hierarchical representation, which motivate the using of e-combs representation:

1. Hierarchical representation treats only agreement between agents, i.e. it enables to represent plans which should be jointly taken by a sub-team. However, it does not enable more complex coordination like, for example, concurrence constraints between agents — two different plans should be operated by two agents concurrently.

2. Hierarchical representation is limited to a strict structure. It does not enable, for example, an agent services in two different sub-teams under some circumstances.

E-combs, however, enable flexible structures with general coordination relations between agents. In addition, while hierarchy is limited to only representing hierarchical organizations, e-comb can represent any coordination between teammates including non-hierarchical organizations like in the shop example.

First, we will briefly define the plan-decomposition hierarchy, and a team organization hierarchy (these have been fully described in [12]). A team organization hierarchy is used to represent a monitored agents' role. All the agents in the system construct a *group*. This group is divided into one or more *subgroups*. Thus, for example, the group in figure 12 is divided into four subgroups: the *Midfielders*, the *Defenders*, the *Forwards* and the *Goalies*. Pay attention, that this is a simplified example, a real system may be fur-

ther divided into *subsubgroups* and so on, where the leaves of the structure tree are the agents themselves.

A plan-hierarchy is used to represent a monitored agent's plan. It is defined to be a directed connected graph, where vertices are plan steps, and edges signify the hierarchical decomposition of a plan into sub-plans. Each of those groups and subgroups has a set of group-plans in which it may be found at any time. For example, Figure 13, presents a portion of the plan-hierarchy used to monitor the ISIS'97 RoboCup Simulation team [11]. The whole group always selects the general plan `WinGame`. Two particular plans are defined for the group, in which it may select when 'winning game' — those are `Play` and `Interrupt`. Each of those is still a group-plan which is applied to all the agents in the system. Under those plans, each of the subgroups has its own possible plans. For example, when the system is executing the `Play` plan, the *Forwards*' plan should be `Attack`, while the *Goalies*' plan should be `Defend`. Dividing into subgroup plans, in this figure, is noted by dashed-line arrows, while solid-arrows represent various options for the same group or subgroup.

Last, when a subgroup selects some subgroup-plan, the agents which it consists of may be in one or more agent-plans. Still in figure 13, we can see that the `SimpleAdvance` plan is connected to the agent-plans (noted as borderless nodes) `ScoreGoal`, `KickOut` and more. That means that the agents of this subgroup must be in one of those plans.

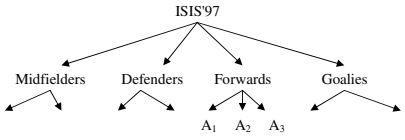


**Figure 12: Teams (groups) hierarchy.**

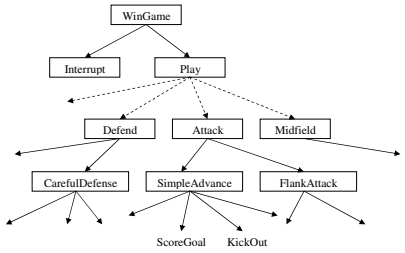


**Figure 13: Plans (states) hierarchy.**

Now, we will show a way to translate the hierarchical plan structure into a *rule* of e-combs (presented in Section 5.1). First, we will present the rationale of the process, then, the actual algorithm, and, at last, demonstrate that in our specific example.

When examining the meaning of the plan hierarchy, we conclude the following understandings. Any plan that is 'splitted' to a few plans which are to be applied *on the same subgroup* (represented by a solid edge), provides, in fact, a 'choice node'. That is, the subgroup must select only one of the splitted plans. In other words, this matches the `OR` operator. On the other hand, any plan that is splitted to a few plans where each of those plans is to applied for a different subgroup (represented by a dashed edge), dictates, in fact, the exact plan in which this subgroup should select, leaving it no choices. In other words, *all* the splitted nodes must be executed (each by a different sub-team). This matches the `AND` operator. At last, any *agent-plan* (noted here as a borderless node) should be applied to each of the agents in the sub-team that points to this plan.

It is worth to mention, that a subgroup node which points to agent-plans is in fact equivalent to a more expressed form, which treats each agent as a one-agent-subgroup. Each of those agents is allowed to be (in service of the particular subgroup-plan) in one of the pointed agent-plans. This situation may be defined by a single e-comb, in which for all the agents in this subgroup only the pointed agent-plans are on ('1'), and for all other agents, *all* the plans are on (rows of 'all ones'—that means ' don't care'). The algorithm itself appears in Algorithm 1.

**Algorithm 1** PlansToRule(plans-tree $T$).

Return a rule of e-combs representing the given plans-tree.

```
for all node i in T do
  if i's offsprings refer to the same group as i then
    replace i with an OR node
  else if i's offsprings refer to different subgroups then
    replace i with an AND node
  else if i's offsprings are agent-plans then
    replace i with an e-comb node, in which all the agents of this subgroup have
    only the pointed plans '1' while the other plans '0', and all other agents' plans
    are all ones
  end if
end for
return T
```

Now, let us demonstrate the algorithm using Figures 13 and 14. The root is a node which its offsprings are still related to the whole group (just like the root itself), hence it becomes an `OR` node. Then, the `Interrupt` and the `Play` nodes are nodes which each of their offsprings are related to a different subgroup. Hence, those nodes become `AND` nodes. The nodes in the next level `Defend`, `Attack`, `Midfield`, etc. point to other nodes which are related to the same subgroup. For example, the node `Attack` is related to the Forwards group, and so are its offsprings, `SimpleAdvance` and `FlankAttack`. Hence, all of those nodes become `OR` nodes. Last, the nodes in the next level e.g., `SimpleAdvance` point to agent-plans. Hence, we should replace them with an equivalent e-comb. For example, the e-comb $C_{\texttt{SimpleAdvance}}$ has the value of one for the relevant sub-team members (the 'forwards', which includes $A_1$, $A_2$ and $A_3$) only in the plans of `ScoreGoal` and `KickOut` (and possibly for other pointed plans). For each agent of other teams the e-comb includes 'all ones' rows. The rule tree is presented in figure 14.

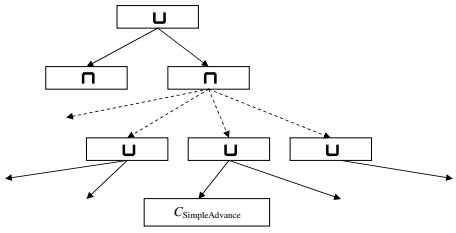


**Figure 14: Rule tree.**

## 7. ADDING DYNAMICS

Until this point we defined the states in which each agent is found at a given time. A more complex system may define 'dynamics' rules. A movement rule means, that if an agent was in some state $s_k$ at time $t-1$, it may only be in a predefined subset of states at time $t$. For example, in the shop system, we may define several such rules:

$M^{8\times 8} =$

| $from \backslash to$ | BREAK | IDLE | NEGOTIATE | SELL | INNERTALK | WATCH | GUARD | EQUIP |
|---|---|---|---|---|---|---|---|---|
| BREAK | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| IDLE | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| NEGOTIATE | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| SELL | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| INNERTALK | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| WATCH | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| GUARD | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| EQUIP | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

**Figure 15: States transition matrix.**

1. An agent may SELL only after NEGOTIATE.

2. An IDLE state will not occur after a BREAK; if the agent has nothing to do, it will continue its BREAK.

3. An INNERTALK will only appear after WATCH, NEGOTIATE, GUARD or EQUIP, or as a continue of a former INNERTALK state.

We may define as many as $s^2$ binary rules of this kind; from each state to each state. We can use a matrix to express them. For example, the above set of rules may be represented by the matrix presented in Figure 15. If an agent is observed TALK, according to the interpretation matrix in Figure 4 it should be now in one of the states {BREAK,NEGOTIATE,INNERTALK,WATCH}. However, if we know that its former state was not one of {NEGOTIATE,INNERTALK,WATCH,GUARD,EQUIP} (the assigned states in column five) — we can be sure that the agent is currently not in INNERTALK state, but in the middle of another state — for example, a BREAK.

In order to use the states transition matrix, we must always keep a track of the interpreted agents' states at time $t-1$. Suppose that the matrix given as the product between the observation matrix at time $t-1$ and the interpretation matrix, is the e-comb $\Omega_{t-1}$ (a kind of such a matrix is presented in Figure 6). In the same way $\Omega_t$ represent the product matrix at time $t$.

In order to infer the states that the agents could be at time $t$ from agents' states at time $t-1$ ($\Omega_{t-1}$), we could use the state transition matrix, by calculating $H_t = \Omega_{t-1} \times M^T$. The multiplication is done in a way of binary matrix multiplication. Meaning, in the same way of the usual manner of matrix multiplication, only that scalar multiplication is replaced with logical 'and' and scalar addition is replaced by logical 'or'. The e-comb $H_t$ (pay attention that its size is still $n \times m$) states, for each agent, what are the *possible* states in which it may be now. In other words, it holds all the states that the agent can move to at time $t$ from the states it has been in for the time $t-1$.

Now we should compare $H_t$ with the interpreted states matrix $\Omega_t$ given by the observation. We compare between them using logical 'and' $F_t = \Omega_t \wedge H_t$. This 'and'ing may reduce the hypothesized states at time $t$, since if, for example, the state in $\Omega_{t_{ij}} = 1$ but $H_{t_{ij}} = 0$ then in the final matrix $F_{t_{ij}} = 0$.

# 8. SUMMARY AND FUTURE WORK

In this paper we presented a new formal approach to team coordination representation. 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. The e-comb is a compact way to represent multiple agents' coordination in one structure. We showed that the matrix-based structure enables an easy and intuitive way to define flexible and scalable coordination between teammates. In addition, this structure enables the using of the normal operations and attributes of matrices, which yields interesting information on the agents. Based on this representation we presented an efficient observation-based fault detection algorithm. The space and time needed for this algorithm are mainly dependent on the complexity of the rule—how many e-combs are involved and in what kind of relations, and not on the team size.

This research is novel in that it presents a general and efficient solution that eases the design of coordination requirements and allows modularity and reuse of already existing systems.

In the future we plan to add partial observation 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 in a system that is cheap in resources, yet very reliable. In addition, in this paper we have presented a dynamic method for the states transition. But, we have assumed 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 representation and algorithm to dynamic coordination.

# 9. REFERENCES


[1] 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.

[2] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group actions. *Journal of Artificial Intelligence Research*, 86:269–358, 1996.

[3] Bryan Horling, Victor R. Lesser, Regis Vincent, Ana Bazzan, and Ping Xuan. Diagnosis as an integral part of multi-agent adaptability. Technical Report CMPSCI Technical Report 1999-03, University of Massachusetts/Amherst, January 1999.

[4] Nicholas R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence Journal*, 75(2):195–240, 1995.

[5] Meir Kalech and Gal A. Kaminka. Diagnosis of multi-robot coordination failures using distributed csp algorithms. In *American Association for Artificial Intelligence (AAAI-06)*, 2006.

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

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

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

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

[10] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[11] Milind Tambe, Gal A. Kaminka, Stacy C. Marsella, Ion Muslea, and Taylor Raines. Two fielded teams and two experts: A robocup challenge response from the trenches. volume 1, pages 276–281, August 1999.

[12] Milind Tambe, David V. Pynadath, Nicholas Chauvat, Abhimanyu Das, and Gal A. Kaminka. Adaptive agent integration architectures for heterogeneous team members. pages 301–308, Boston, MA, 2000.