A Representation for Coordination Fault Detection in Large-Scale Multi-Agent Systems

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

August 2, 2009

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 faults, 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 efficient resolution for large-scale teams. 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. We demonstrate the modularity of the representation by presenting a reuse of existing systems and by importing other models (e.g. hierarchical systems) into the new representation. Finally, we extend the representation to support dynamical aspects of complex systems.

*This research was supported in part by ISF grant #1357/07.

1 Introduction

Autonomous agents within multi-agent systems interact and coordinate to achieve their goals [8, 5]. The representation of the coordination is a key requirement since it may influence the complexity, modularity and flexibility of reasoning operations of the team [28, 12].

One aspect of this requirement is detecting coordination faults. The increased deployment of robotic and agent teams in complex dynamic settings has led to an increasing need for coordination faults handling [18, 15, 7, 10]. Coordination-fault detection does not indicate whether the group is achieving its goals but only if agent-coordination exists. Detection of coordination faults is essential for a recovery process (e.g., a negotiation) during which cooperation is reinstated.

A demonstration of the strong need in fault detection is provided in the following example, which we use throughout this text. A crew of agents is responsible for the operation of a shop. In order for the shop to funcion as required, several conditions must exist. For example, some agents must serve customers; one agent must always guard; agents might take a break, as long as the other tasks are being handled, and so on. Each of the agents can choose what to do at any time, according to its role. Thus, it is possible that each of the agents is acting legitimately, and still, the system as a whole is broken. For example, it might happen that the shop is left unguarded, or that customers are not being served. A fault-detection mechanism can identify such a case. Then, it is possible to recover the system, e.g., by forcing one of the agents to immediately start serving customers.

The ability to detect faults is directly affected by the representation of the coordination. A näive approach, for example, which maintains and reasons about the whole coordination space is computationally time consuming, and/or takes up a lot of space [13, 1]. The motivation for this paper is to find a simple, efficient and modular representation which will enable to represent the coordination easily.

Previous coordination-faults detection methods have adopted an approach of using a causal-model which describes a pre-defined set of faults, i.e. a set of forbidden coordination relations [15, 7]. This approach is not complete, since it does not guarantee that all the possible faults are pre-defined. In addition, it requires the designer to spend time on the research of possible faults, rather than focus on the required coordination. Others are only able to capture specific coordination faults, such as disagreements [13, 12].

The drawbacks of these approaches are caused by the representation model of the coordination. On the one hand, works that tried to find a general solution for the problem [13, 7], provided mechanisms with exponential complexity of time

or space. On the other hand, works that tried to overcome the complexity issue, provided solutions for specific types of faults or for specific team models, e.g., hierarchical teams [12]. None of the works have taken a systematic approach to addressing this challenge. In this work, we suggest a representation that is general, in the sense that it fits any team model and any type of fault, yet has a low, polynomial complexity. We do that by modeling the normal behavior of a team, that is, the permitted coordination among the agents.

We present an efficient Boolean matrix-based approach to represent: (1) predefined 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, we do not represent the relations between teammates explicitly, but gather them compactly in matrix structures. Therefore, this approach is scalable in the number of agents and states. Finally, the use of a matrix-based representation enables the use of the matrix operations and yields interesting information about the agents.

This research takes advantage of the fact that many different specific joint-states might be expressed by one, more general, definition. We use Boolean matrices for this kind of general definition. In order to provide flexibility in the design, and allow any type of required coordination, we use special operators. Several matrices might be combined using those operators to construct complex coordination rules. These operators also open the way for an easy reuse of representations of existing systems as parts of larger systems.

The representation we suggest also facilitates easy definition of dynamic systems. In these systems, progression of the system at run-time affects its definitions. We introduce two kinds of such dynamics. First, the representation and evaluation of temporal rules, that dictates the states an agent is allowed to choose. Second, an efficient definition of dynamic systems, in which the allowed coordination is changed according to external conditions. Overall, the matrix representation enables an easy and efficient way to implement functions on team, like plan recognition, fault detection, diagnosis and other reasoning operations.

To demonstrate the new representation, we will present a fault-detection algorithm based on the matrix representation. The algorithm (described in Section 4) uses matrix operations to efficiently find coordination faults. The algorithm can, in many cases, reduce the complexity of detection in large-scale teams from exponential to polynomial. In the rest of the text we extend the usage of this

representation to additional domains and systems. We analytically explore the computational implications of using this representation, and the failure-detection capabilities provided by the algorithms presented.

The paper is organized as follows. Section 2 presents related work. In Section 3, we present our new representation of the coordination and observations of others, 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. Section 8 summarizes.

2 Related Work

Some works address coordination faults in teams of agents, but most of them do not consider efficiency aspects for large-scale teams. Other models require a predefined list of all the possible faults which, in terms of coordination fault, grows exponentially in the number of agents.

The most related work is of Kaminka at al. [13, 1]. They 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 [12] and later Kaminka [**?**] present a scalable method for such assessment. Our analytical results and formulation generalize the results in [13].

Based on the hierarchical model, Kalech and Kaminka [11] propose a diagnosis for disagreement failures, which is step beyond the fault detection. They even extend their method to large-scale teams [9]. Unfortunately, the drawback of all these works is that they require detection of system faults in cases where the desired joint states are those of perfect agreement. In addition, their hierarchical representation of the coordination restricts the organizational relations between the agents.

Williams et al. [31, 14] provide a model for cooperation of unmanned vehicles. They coordinate these vehicles by introducing a reactive model-based programming language (RMPL). This model is robust and so it could detect faults and recover. However, their model-based language addresses only small systems.

Dellarocas and Klein [15, 2] report on a system of domain-independent exceptions handling services. A first component contains a knowledge base of generic exceptions. A second component contains a decision tree of diagnoses; the fault detection process is done by traversing down the tree by asking queries about the relevant problem. A third component is responsible for seeking a solution to the exception, based on a resolution knowledge base. This approach transfers the fault-handling responsibility from the agent to an external system, to alleviate the load on each agent designer (which would now be freed of the responsibility of implementing an exception-handling system in each agent).

Similarly, Horling et al. [7] use a fault-model of failures to detect and respond to multi-agent faults. In this model a set of pre-defined possible faults are stored in acyclic graph's nodes. When a fault is detected a suitable node is triggered and according to the fault characters the node activates other nodes along the graph. The advantage of Horling's fault-model system over Dellarocas and Klein's system is the use of a learning algorithm that can be employed to maintain structure as time passes. As with Dellarocas and Klein, in Horling's work scale-up concerns are not addressed. In addition, their fault-model approach dictates that all possible faults be analyzed in advance.

Based on the fault-model approach, Pencolé et al. [19] and Lamperti and Zanella [16] use a discrete-event system [25, 24] to model a distributed system where the possible faults are modeled in advance. The diagnoser infers unobservable faulty events by computing possible paths in the discrete event system that match observable events. A common theme in all of these is that they require pre-enumeration of faulty interactions among system entities. However, in multi-agent systems, these are not necessarily known in advance since they depend on the specific run-time conditions of the environment, and the actions taken by the agents.

Poutakidis et al. [20] provides a method for tracking the progress of conversations using interaction protocols, and detection of some faults, using a Petri-net representation of the interaction protocols that are expected to take place (rather than the expected faults 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 [6].

Some works address diagnosis of multi-agent systems which is a related issue to fault detection, but none of them consider the problem of large-scale teams. For instance, Fröhlich et al. [4] and Roos et al. [21, 22, 23] suggest dividing a spatially distributed system into regions, each under the responsibility of a diagnosing agent. If the fault depends on two regions the agents that are responsible

for those regions cooperate in making the diagnosis. However, the interactions among system entities are not necessarily known in advance since they depend on the specific conditions of the environment at runtime, and the appropriate actions assigned by the agents [17]. It is impossible to address this by keeping all the possible interactions between the agents; this might increase communication complexity, especially in large systems, since the number of candidate diagnoses is exponential in the number of dependencies.

A number of previous works address scalability in multi-agent systems, but do not consider diagnosis. Scerri et al. [26, 27] address tasks of team coordination among the members of large teams. Specifically, they developed algorithms meeting the requirements of large teams for planning, sharing information and task allocation—but not fault-detection. They achieve the scalability by organizing all members into an associated network. The associated network is performed at the initialization and remains static during the execution. In this paper we propose coordination rules that are dynamically changed.

Durfee [3] discusses heuristic methods for reducing the knowledge that agents use in coordination. The methods are based on hierarchies and abstractions which depend on the task environments and collective behavior of the team. Like the former work, this work also addresses large-scale teams but does not consider the fault-detection problem.

This paper presents a formalization to represent team coordination based on matrix structures. This representation enables a systematic approach to coordination faults detection 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 fault. 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. This could be suitable to large-scale teams.

3 Fundamental Objects

This section presents the basic objects used in multi-agent systems and the relations among them. Before that, we will present some general algebraic notations we will use throughout the text. Some important structures we use here are Boolean matrices. We notate a Boolean matrix of order $i \times j$ as a matrix in $\mathbb{B}^{i \times j}$. We define the following logical operators:

Definition 3.1 (Boolean matrices logical operators). *Let M, N be matrices of order $p \times q$ over the Boolean space. Then*

$$\begin{aligned} M \wedge N &= T \in \mathbb{B}^{p\times q}, \text{ such that } t_{ij} = m_{ij} \wedge n_{ij} \\ M \vee N &= T \in \mathbb{B}^{p\times q}, \text{ such that } t_{ij} = m_{ij} \vee n_{ij} \end{aligned}$$

In addition, we will define matrix product, in a similar way to the usual meaning of it:

Definition 3.2 (Boolean matrices product). *Let $M \in \mathbb{B}^{p\times q}, N \in \mathbb{B}^{q\times u}$ be two Boolean matrices, then their product $D = M \cdot N$ is in $\mathbb{B}^{p\times u}$ and is defined to be*

$$D : d_{ij} = \bigvee_{k=1}^{q} m_{ik} \wedge n_{kj}$$

This is in fact very similar to the usual matrix product, where the scalar product is substituted by logical-AND and the scalar summation is substituted by logical-OR.

Now, we can move on to the specific structures of multi agent systems. 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 (we will refer this example as "the shop "): ANNY the manager, BENNY the cashier, two sellers — CANNY and DANNY, ERNY the storekeeper and a guard, FRENNY:
A_{shop}={ANNY, BENNY, CANNY, DANNY, ERNY, FRENNY}
Agents may be in one of eight possible states:
S_{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 3.3 (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 3.4 (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 $\langle \text{agent}, \text{state} \rangle$ pairs. Any of these pairs may define a single state in which an agent is found, as we will describe later. A special yet common case is, when the state of an agent is not defined. In other words, its state can be any of the states. In this case, we notate its state with the 'don't care' symbol, '$\varnothing$'. For this purpose, we extend the above to a 'complete form':

Definition 3.5 (Complete-environmental-space). *Let $E = \langle A, S \rangle$ be an environment. The Cartesian product $A \times (S \cup \{\varnothing\})$ is called the* complete environmental space *of E, and is denoted $E^{\oplus}$.*

In addition to pairing each agent to one state (or to none of them), we would like to define a space which allows each agent to be paired with any set of states (including the empty set). For that purpose, we define the following space:

Definition 3.6 (Complete-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* complete environmental power space *of E, and is denoted $E^{\otimes}$.*

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 3.7 (Position). *Let $E = \langle A, S \rangle$ be an environment. A pair $\langle a', s' \rangle \in E^{\oplus}$ 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 3.8 (Superposition). *Let $E = \langle A, S \rangle$ be an environment. A pair $\langle a', S' \rangle \in E^{\otimes}$ (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 joint-state [13, 1], or *coordination*, among the agents. In order to save the generic nature of it, we define the coordination as a function.

Definition 3.9 (Coordination). *A* coordination *function over an environment* $\langle A, S\rangle$ *is a function that* positions *each agent in a particular state or in no state:*

$$\gamma : A \to (S \cup \{\varnothing\}) .$$

If all the agents in γ *are mapped to states in* S*, the coordination is said to be* full*. Otherwise—that is, if one or more agents are mapped to the* $\varnothing$*— the coordination is* partial*.*

Coordinations are the essense of multi agent systems. At every moment, each agent is found in a specific state, which means that at every moment the agents are found in a specific full coordination. The designer of the system is in charge of defining the allowed and disallowed coordinations. The set of all allowed coordinations is called *policy*.

Definition 3.10 (Policy). *Let* E *be an environment. A* policy *over* E *is a function*

$$\varphi : \{\textit{coordinations over } E\} \to \mathbb{B}$$

(where $\mathbb{B}$ *is the Boolean space). For each coordination* γ *over* E*, the policy defines whether it is* legal *(*$\varphi(\gamma) = 1$*), or* illegal *(*$\varphi(\gamma) = 0$*). We notate the subset of all legal coordinations* $\varphi_{\textit{legal}}$*, and the subset of all illegal coordinations* $\varphi_{\textit{illegal}}$*.*

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

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 3.8).

Definition 3.11 (Supercoordination). *A* supercoordination *function (or s-coord for short) over some environment* $E = \langle A,\ S\rangle$ *is a function*

$$\Gamma : A \to \|S\|$$

i.e., it positions each agent in a set *of possible states.*

$$\Gamma(a) = \begin{cases} \{\text{INNERTALK, WATCH}\} & a = \text{ANNY} \\ \{\text{BREAK, SELL}\} & a = \text{BENNY} \\ \left\{ \begin{matrix} \text{BREAK, NEGOTIATE,} \\ \text{SELL, EQUIP} \end{matrix} \right\} & a \in \{\text{CANNY, DANNY}\} \\ \{\text{GUARD}\} & a = \text{ERNY} \\ \{\text{BREAK, INNERTALK}\} & a = \text{FRENNY} \end{cases}$$

Figure 1: A supercoordination function.

We would now like to use an algebraic representation for the s-coords. Moving to the algebraic realm naturally allows a variety of calculations and manipulations on those structures, as described later. Assume an agent set $A = \{a_1, a_2, \ldots, a_n\}$ and a state set $S = \{s_1, s_2, \ldots, s_m\}$. We can represent a supercoordination of the agents by a Boolean matrix of order $n \times m$. This matrix represents a combination of the agents' superpositions:

Definition 3.12 (Supercombination). *Let E be the environment $\langle A,\ S\rangle$, where $|A| = n$ and $|S| = m$, and let Γ be an s-coord over E. A* supercombination *(or s-comb for short) C over E is a Boolean matrix of order $n \times m$ ($C \in \mathbb{B}^{n\times m}$). The supercombination-representation of Γ provides:*

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

Figure 1 presents an example for such a function, and Figure 2 presents its appropriate s-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. For example, regarding the two first agents, one coordination constraint could be "ANNY selects state INNERTALK while BENNY selects SELL". Another coordination that is represented by this s-comb is "ANNY selects 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 s-comb it will be represented by a row of zeros.

Definition 3.13 (Ill-supercoordination). *Let Γ be a supercoordination function over some environment $E = \langle A,\ S\rangle$. Then Γ is an* ill-supercoordination *iff $\exists\ a \in A \mid \Gamma(a) = \emptyset$. Using the s-comb representation: $\exists i,\ 1 \leq i \leq n :\ \bigvee_{j=1}^{m} c_{ij} = 0$. S-coords or s-combs that are not ill, are said to be* vital.

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

Figure 2: The s-comb representation of the supercoordination appears in Figure 1.

The example shown in Figure 1, for instance, is of a vital-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 logical states 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 3.14 (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 λ_a, the function λ_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.

Definition 3.15 (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 $\Lambda_a : B \to \|S\|$.*

Λ_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 3.16 (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:*

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

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, in many cases, infeasible, since it involves massive communication resources. Our observation-based approach

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

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

$$\Theta^{6\times 9} = \begin{array}{c} \\ \textsc{Anny} \\ \textsc{Benny} \\ \textsc{Canny} \\ \textsc{Danny} \\ \textsc{Erny} \\ \textsc{Frenny} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \textsc{Talk} & \textsc{Phone} & \textsc{Stand} & \textsc{Walk} & \textsc{Counter} & \textsc{Put} & \textsc{Get} & \textsc{Carry} & \textsc{Other} \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 5: An observation matrix.

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

Definition 3.17 (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 $\omega : A \to B$, that maps each agent to a particular action. Θ stands for the observation matrix representation:*

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

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

After introducing the fundamental entities, we can formally define a whole multi agent system.

Definition 3.18 (System). *A* system *is a 5-tuple* $\mathbb{S} = \langle A, S, B, \lambda, \varphi \rangle$*, where*

- A *is a set of* n *agents,* $\{a_1, a_2, \ldots, a_n\}$*,*
- S *is a set of* m *states,* $\{s_1, s_2, \ldots, s_m\}$*,*
- B *is a set of* ℓ *actions,* $\{b_1, b_2, \ldots, b_\ell\}$*,*
- λ *is a set of* n *latitude functions,* $\{\lambda_{a_1}, \lambda_{a_2}, \ldots, \lambda_{a_n}\}$*, and*
- φ *is a policy function over* $\langle A,\ S \rangle$*.*

If the latitude functions of all the agents are identical, the system is said to be homogeneous*, and we simply refer the latitude as* λ*.*

A system, thus, consists of all the relevant entities (*agents*, *states* and *actions*) and of the logic to which these entities are obliged (*latitude* and *policy*).

4 A Case Study — Fault Detection

In the former section we defined a formalism to a matrix-based representation for team coordination. We defined a supercombination matrix between the agents' states (s-comb, Definition 3.12), an interpretation matrix for inferring the current states of the agents by their actions (Definition 3.16) and an observation matrix which maps between the agents and their current observed actions (Definition 3.17). 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 3.7) do not match the expected coordination given by the policy (Definition 3.10). Using Definition 3.12, we can represent the policy using a Boolean matrix, say, C. 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: $\Omega = \Theta \cdot I$ (Θ is the observation matrix and I is the interpretation matrix), where Ω is an $n \times m$ Boolean matrix (that is, an s-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 Θ by element k in column j of I. This multiplication, of course, is '1'

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

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

iff both of them are '1'. Since each row in Θ has exactly one element which is '1', the value of each element in Ω will be at most '1'.

For example, Figure 6 presents the s-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.

We can now explain the fault detection algorithm. A fault is defined as a situation wherein none of an agent's possible assigned state (according to Ω) appears on the 'legal coordination' of the policy, designated as C (the policy s-comb). In order to examine possible matches we will operate a logical 'and' between C and Ω 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 s-comb.

R represents all the agents-assigned coordinations that satisfy C according to interpreted states by the observation. The coordinations represented by R are all those that positions each agent a_i in one of the states s_j, that match '1' elements 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 coordination 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 3.13), meaning, an all-zero row exists, then the assigned agents' states are definitely forbidden. In this case, a fault alert is warranted. This operation takes only $O(nm)$ operations (counting the '1's for m elements on each of R's n rows).

The algorithm is presented in Algorithm 1. It takes two input s-coord arguments—the policy and the observation. First, it calculates the logical-AND between them, and assigns the result to R (line 1). Then, it starts searching R. It searches all rows from 1 to n (line 2). Within each row, it searches all columns

from 1 to m (line 4). Whenever an element of '1' is found, the algorithm breaks the column search (line 7) and continues to the next row (in line 2). If the column search ends without any '1' element found, the algorithm return FAULT (line 11). If the search of the s-comb is finished without finding any all-zero row, the algorithm returns NOFAULT (line 14).

Algorithm 1 TestObservation(observation s-coord Ω, policy s-coord C).

Test whether Ω (the state interpretation of an observation) violates the policy defined in C. Returns FAULT if it is, or NOFAULT otherwise. Both input s-coords are of order $n \times m$.

```
1:  R ← C ∧ Ω
2:  for i ← 1 to n do
3:      flag ← false
4:      for j ← 1 to m do
5:          if r_ij = 1 then
6:              flag ← ture
7:              break
8:          end if
9:      end for
10:     if flag = false then
11:         return FAULT
12:     end if
13: end for
14: return NOFAULT
```

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 Ω (Figure 6). In this s-comb, the two bottom lines, representing ERNY and FRENNY, are all-zero. When the algorithm finishes searching row 5, no '1' element is found, and FAULT is returned. This means that no desired combination can explain the agents' actions. A fault has been detected.

In order to detect faults by observations only, we define two approaches of decision [13] (the term in [13] is *policies* rather than *approaches*, but this text reserves the term for the policy function, φ). The *optimistic approach* assumes that as long as the system is not proven to be faulty, no fault should be reported. Using this approach, one can never get a false alarm. If it reports a fault, then a fault has certainly occurred. The other approach is the *pessimistic approach*. This approach reports a fault in the system, unless it is completely confident that no fault has

$$R = \Omega \wedge C = \begin{array}{c} \\ \textsc{Anny} \\ \textsc{Benny} \\ \textsc{Canny} \\ \textsc{Danny} \\ \textsc{Erny} \\ \textsc{Frenny} \end{array} \begin{array}{c} \begin{array}{cccccccc} \textsc{Break} & \textsc{Idle} & \textsc{Negotiate} & \textsc{Sell} & \textsc{InnerTalk} & \textsc{Watch} & \textsc{Guard} & \textsc{Equip} \end{array} \\ \left(\begin{array}{cccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}$$

Figure 7: The s-comb given by Boolean 'and' operation between the desired coordination C and the interpretation s-comb Ω.

occurred. Using this approach, one can never get to a situation of an unreported fault. We have adopted here an optimistic approach, thus in matrix Ω 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.

We prove the capabilities of the optimistic approach analytically below. To prove that Algorithm 1 detects only coordination faults (i.e., never has false positives), let us prove first that the optimistic approach infers all the possibilities of the states that could be taken by the observed agents. Formally:

Lemma 1. *Suppose we are given an observation matrix I and an interpretation matrix Θ. If $\omega_{ij} = 0$, where ω_{ij} is a cell in the product matrix $\Theta \cdot I = \Omega$, then this entails that agent a_i could* **not** *take state s_j.*

Proof: ω_{ij} is a result of the vectorial product between the i'th row vector in matrix Θ and the j'th column vector in matrix I. $\omega_{ij} = 0$ implies that for each $k \in \{1, 2, ..., m\}$ either $\theta_{ik} = 0$ or $i_{kj} = 0$. If $\theta_{ik} = 0$ then agent a_i does not take the action b_k, thus all the associated states with action b_k in matrix I could not taken by agent a_i. If $\theta_{ik} = 1$ then the associated states with b_k could be taken by agent a_i, however, since $\theta_{ik} = 1$, $i_{kj} = 0$, thus state s_j could not be taken by agent a_i.□

Based on this Lemma we can prove the soundness of our algorithm. Formally:

Theorem 1. *Algorithm 1 is sound.*

Proof: To show that the algorithm is sound, we must show that whenever a row of zeros exists in matrix R, necessarily entails a coordination fault has occurred.

$$\Theta^{6\times 9} = \begin{array}{c} \\ \textsc{Anny} \\ \textsc{Benny} \\ \textsc{Canny} \\ \textsc{Danny} \\ \textsc{Erny} \\ \textsc{Frenny} \end{array} \begin{array}{c} \begin{array}{ccccccccc} \textsc{Talk} & \textsc{Phone} & \textsc{Stand} & \textsc{Walk} & \textsc{Counter} & \textsc{Put} & \textsc{Get} & \textsc{Carry} & \textsc{Other} \end{array} \\ \left(\begin{array}{ccccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}$$

Figure 8: An observation matrix used in proof of Theorem 2.

This will mean that the algorithm never reports failures unnecessarily (i.e., never has false positives.

Without loss of generality, assume the cells in row 1 in matrix R are zero: $\forall j \in \{1, 2, ..., m\}\ r_{1j} = 0$ and let us prove that there is coordination fault. $\forall j \in \{1, 2, ..., m\}\ r_{1j} = 0$ entails that $\forall j \in \{1, 2, ..., m\}$ either $c_{1j} = 0$ or $\omega_{1j} = 0$. If $c_{1j} = 0$ then we do not expect agent a_1 to take state s_j, and thus we could not infer coordination fault. However, based on definition 3.8 $\exists j \in \{1, 2, ..., m\} | c_{1j} = 1$, that says that agent a_i must take one state s_j. $\forall j | c_{1j} = 1 \Rightarrow \omega_{1j} = 0$. However, based on the above Lemma, $\omega_{1j} = 0$ entails that agent a_i could **not** take state s_j. That is a coordination fault.□

We have shown that algorithm 1 is sound, meaning, a row of zeros in matrix R necessarily entails coordination fault. However, it is not complete. In other words, it is possible that although there is a coordination fault, matrix R does not contain a row of zeros.

Theorem 2. *Algorithm 1 is not complete.*

Proof: To prove the incompleteness of the algorithm it suffices to present a counter example in which a failure occurrs, yet the a row of zeros does not exist in matrix R.

Assume the coordination matrix C as in Figure 2, but the actual states taken by the agents are $\{\langle \textsc{Anny}, \textsc{InnerTalk}\rangle$, $\langle \textsc{Benny}, \textsc{Break}\rangle$,$\langle \textsc{Canny}, \textsc{Break}\rangle$, $\langle \textsc{Danny}, \textsc{Negotiate}\rangle$,$\langle \textsc{Erny}, \textsc{Watch}\rangle$, $\langle \textsc{Frenny}, \textsc{InnerTalk}\rangle\}$. Obviously, this case contains a coordination fault since agent ERNY takes state WATCH. Assume the observation matrix shown in Figure 8. The product matrix Ω is presented in Figure 9 and the result matrix in Figure 10. There is no row of zeros in R although we set a case with a coordination fault, thus this algorithm is incomplete.□

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

Figure 9: The s-comb given by the product between the observation matrix and the interpretation matrix, used in proof of Theorem 2.

$$R = \Omega \wedge C = \begin{array}{c} \\ \textsc{Anny} \\ \textsc{Benny} \\ \textsc{Canny} \\ \textsc{Danny} \\ \textsc{Erny} \\ \textsc{Frenny} \end{array} \begin{array}{c} \begin{array}{cccccccc} \textsc{Break} & \textsc{Idle} & \textsc{Negotiate} & \textsc{Sell} & \textsc{InnerTalk} & \textsc{Watch} & \textsc{Guard} & \textsc{Equip} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 10: The s-comb given by Boolean ‘and’ operation between the desired coordination C and the interpretation s-comb Ω, used in proof of Theorem 2.

The theorems above generalize previously proven theorems in [13], which discussed the optimistic policy for the case of detecting failures in a specific type of coordination—agreement. In the earlier results, however, the agents could detect failures only in the case where they agree on a specific plan to be selected by all team-membrs. Thus the earlier work has in fact touched on special cases of the theorems above.

5 Complex Coordination

In many cases, one s-comb will not suffice for a full policy definition. Thus, the s-comb we introduced earlier (Definition 3.12), may only partially define the legal coordinations in a policy. For instance, in the the shop example, assume ERNY could replace FRENNY in GUARD duty. The s-comb in Figure 2 does not deal with this new relation. Moreover, we cannot add another state to C, by just changing $c_{6,7}$ $\langle \text{FRENNY}, \text{GUARD} \rangle$ 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 policy so that it consists of more than one s-comb, without becoming exponentially complex.

5.1 S-Combs Operators

The most important operator used to join a few s-combs is the 'or', notated as ' $\sqcup$ '. Defining two sets of coordinations $C_1 \sqcup C_2$, 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 Ω satisfies the property of being vital (Definition 3.13) with *either* C_1 or C_2 (or both, of course), there is no fault. 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 s-combs using operators a *rule*. Testing an observation s-comb Ω 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 s-combs. That is, for each C_k in $\mathcal{R}$, calculating the result matrix R_k by logically 'and'ing Ω with C_k in an element-by-element fashion (mathematically: $R_k = I \wedge C_k$). Then, each R_k illness is checked. Due to the nature of the operator ' $\sqcup$ ', it is enough to verify that at least one such R_k is a vital s-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 s-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 illness property for Ω must hold for *both* C_1 and C_2. In fact, any expression of the form $C_1 \sqcap C_2$, might be reduced to an equivalent s-comb, that represents exactly the same set of combinations. This is the s-comb $C_1 \wedge C_2$ (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 s-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. S-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 policy rules, correspondingly. The first task would be to assemble all agents into one system. Since we joint the agents in the two shops to one large shop, there are now 12 agents. Instead of using s-combs of order 6×8 we use 12×8 s-combs. Then, we must update definitions from both shops from a 6×8 domain to the new unified one. For this purpose, we expand each s-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 s-combs in such a way that they will become rows 7 to 12 of the new s-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 s-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 s-comb interpretation. The algorithm uses a *tree representation* of the rule. The leaves are the rule's s-combs, and the inner nodes are the operators. The algorithm traverses the tree in a bottom-up fashion and unifies s-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 s-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 offspring are already guaranteed translation into images. Let us introduce the types of image:

[s-combs:] These are in fact the leaves of the tree; each s-comb node becomes an image which includes only one s-comb.

['Or' nodes:] Each 'or' node is replaced by an image that includes all the s-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 s-combs from each of the offspring. In other words, if a node has b images offspring, each consisting of c_b different s-combs, then it will be replaced with an image that includes $\prod_{i=1}^{b} c_b$ s-combs. Each of those s-combs is built of a different combination of b s-combs, which are logically 'and'ed in an element-by-element fashion.

Algorithm 2 presents the reduction procedure of a complex rule tree to one image of s-combs. The algorithm obtains the root of the tree and recursively reduce the tree in a bottom-up manner as described above.

Algorithm 2 ReduceTree(node N).

```
1: if N is a leaf then
2:    replace s-comb in N with an image that contains the s-comb
3: else
4:    for all node i in children of N do
5:       ReduceTree(i)
6:    end for
7: end if
8: if N is an 'OR' node then
9:    replace all offspring images by one image I, where I contains all the s-
      combs in the offspring images.
10: else if N is an 'AND' node then
11:   replace all offspring images by one image I, where I contains all the com-
      binations between the s-combs in the offspring images.
12: end if
```

In order to demonstrate Algorithm 2, let us refer to the following rule on some

s-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 Figure 11. The root has four offspring, two of which (the first and the third) are simple s-combs. The rightmost is an 'and' node with three simple, s-combs offspring. The second one, is an 'and' node, with two offspring, themselves sub-trees, each consisting of an 'or' node and two s-combs offspring.

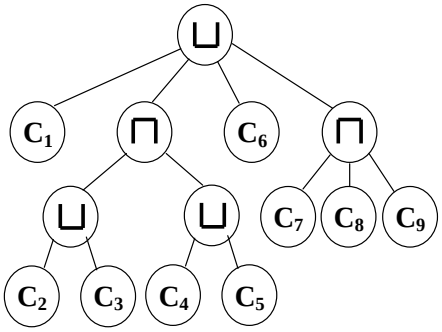

Figure 11: The Rule Tree for $\mathcal{R}$.

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 s-comb. It is therefore replaced by a simple image node that includes exactly this s-comb.

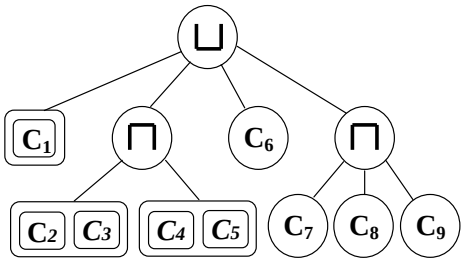

Figure 12: Rule Tree Reduction – step 1.

In the next stage, the same thing is done to the next leaf (the s-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 12).

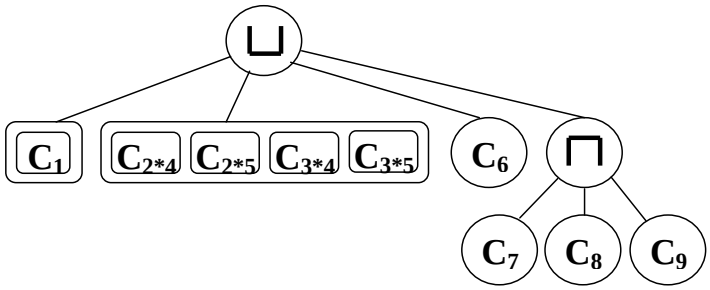

Figure 13: Rule Tree Reduction – step 2.

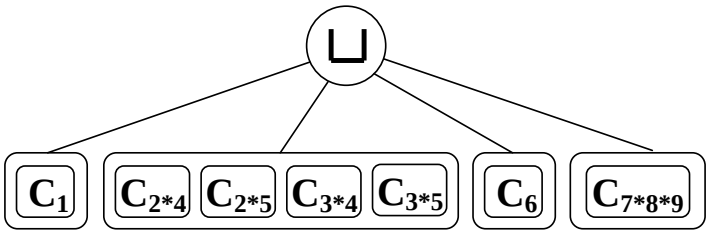

Figure 14: Rule Tree Reduction – step 3.

Next, we have an 'and' node, with two images offspring, each of which consists of two s-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 13). As we have already mentioned, 'and'ing s-combs ($\sqcap$) is in fact identical to an element-by-element 'and'. Hence, each of the expressions $C_{x\star y}$ is one s-comb. During the next stage, the node of C_6 is replaced by an image with only this s-comb. Then the rightmost 'and' node, with three offspring (C_7, C_8, C_9) is replaced with an image of one s-comb, which is the result of 'and'ing those three s-combs — $C_{7\star8\star9}$ (Figure 14). At this stage, we reach the root 'or' node, which has four images offspring.

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

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 s-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. This is unlike other approaches, which are exponential in the number of agents and states.

In order to compute the best and worst-case complexity of the tree reduction process in Algorithm 2, we should find the appropriate best and worst-case tree structures. Let b denotes the branching factor of the tree and h its height, and prove the complexity of the best and worst case of algorithm 2.

Theorem 3. *The best-case complexity of Algorithm 2 is* $O(b^h)$.

Proof: The complex rule operators have the associativity property:

$$(C_1 \sqcap C_2) \sqcap (C_3 \sqcap C_4) = C_1 \sqcap C_2 \sqcap C_3 \sqcap C_4$$

and

$$(C_1 \sqcup C_2) \sqcup (C_3 \sqcup C_4) = C_1 \sqcup C_2 \sqcup C_3 \sqcup C_4$$

Thus, the best-case is a tree that contains only 'AND' or only 'OR' operators. In those cases we can simply reduce the tree by operating the 'AND' or 'OR' operators between the leaf s-comb nodes. Thus, the best-case complexity is $O(b^h)$. □

Theorem 4. *The worst-case complexity of Algorithm 2 is* $O(b^{b^{(h-1)}})$.

Proof: To compute the worst-case, we should clarify again the image computation in the tree. An 'OR' operator just collects all of the s-combs in its offspring images into one image which is $O(|image|b)$, where $|image|$ represents the image size (number of s-combs in the node). However, the 'AND' operator utilizes distribution law

$$(C_1 \sqcup C_2) \sqcap (C_3 \sqcup C_4) = (C_1 \sqcap C_3) \sqcup (C_1 \sqcap C_4) \sqcup (C_2 \sqcap C_3) \sqcup (C_2 \sqcap C_4)$$

which includes all the combinations between any of the s-combs in the b offspring images, which is $O(|image|^b)$. Thus, the tree reduction grows polynomially with 'OR' nodes and exponentially with 'AND' nodes. The worst case tree structure contains all internal nodes are 'AND' except of the last internal level that contains 'OR' nodes (see Figure 15). In this structure the tree reduction grows exponentially in the hight of the tree h and in the branching factor b in a bottom-up manner.

In particular, each 'OR' node in the last internal level includes its b s-comb leaf children. Each 'AND' node in the above level operates over its b 'OR' node children (where each one of them contains b s-comb). The combination size over the b 'OR' children is b^b. In the next above level, the 'AND' node operates over b 'AND' children nodes, where each one of them contains b^b s-combs. The number of combinations is $(b^b)^b = b^{(b^2)}$. Inductively, the complexity continues to grow exponentially in the levels of the tree, where each 'AND' node operates over its b children and increases the number of s-combs exponentially in b. Thus, for hight tree h, the number of s-combs after reducing the tree is $b^{b^{(h-1)}}$, so the worst-case complexity is $O(b^{b^{(h-1)}})$. □

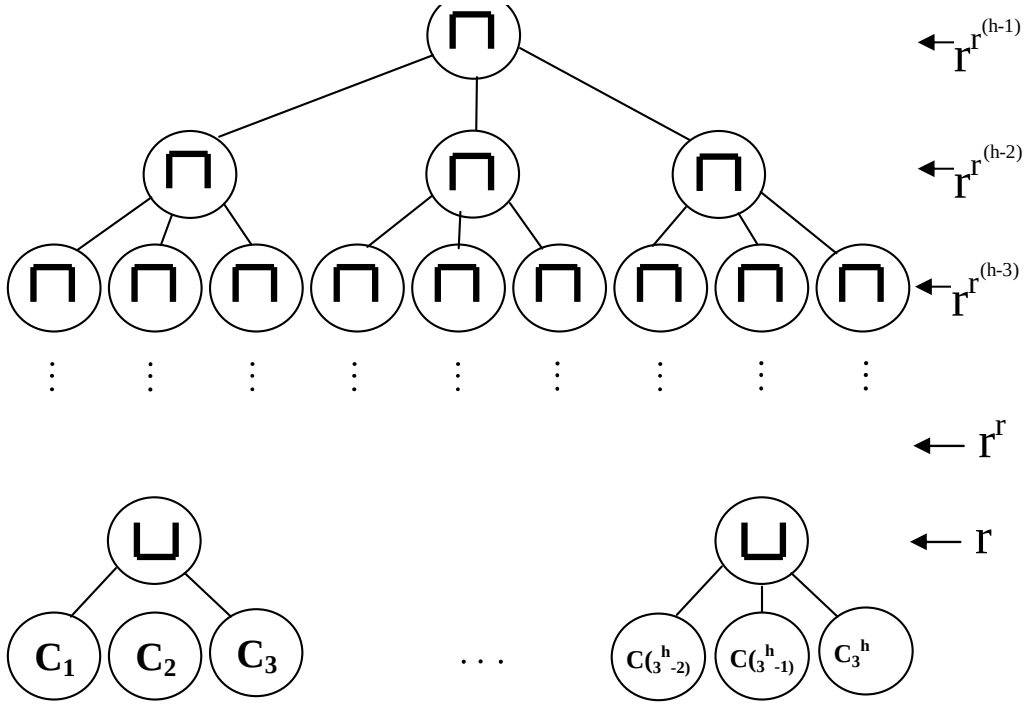

Figure 15: Worst-case complexity structure.

6 Hierarchical Structures

Following the previous section, one of the benefits of using s-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 [28] using the s-combs concept.

We see a few drawbacks in hierarchical representation, which motivate the use of s-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, concurrent 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 serves in two different sub-teams under some circumstances.

The use of s-combs, however, facilitates flexible structures with general coordination relations between agents. In addition, while hierarchy is limited to only representing hierarchical organizations, s-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 [30]). 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 16 is divided into four subgroups: the *Midfielders*, the *Defenders*, the *Forwards* and the *Goalies*. This is a simplified example; a real system may be further 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 17, presents a portion of the plan-hierarchy used to monitor the ISIS'97 RoboCup Simulation team [29]. 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 17, 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.

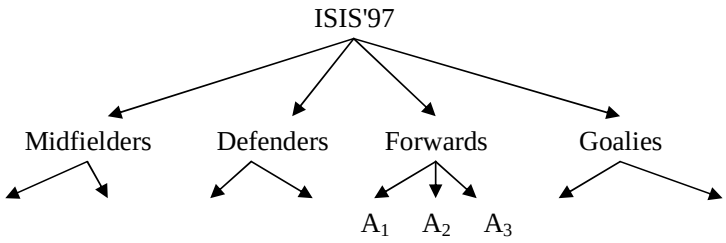

Figure 16: Teams (groups) hierarchy.

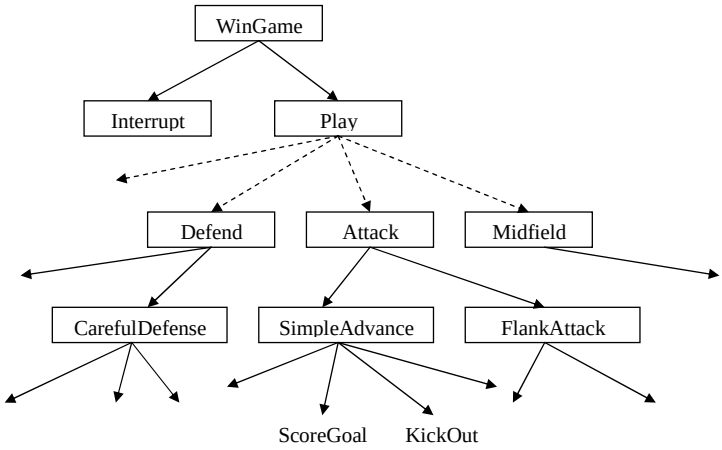

Figure 17: Plans (states) hierarchy.

Now, we will show a way to translate the hierarchical plan structure into a *rule* of s-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 'split' 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 split plans. In other words, this matches the `OR` operator. On the other hand, any plan that is split 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 split 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 s-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 3.

The first line of the algorithm starts a loop over all the nodes in the tree. For each of those node, the type of node is being checked, and an action is taken in accordance. Line 2 detects choice nodes — nodes that are split to several optional plans for the same group. These nodes are replcaced with `OR` nodes (line 3). Line 4 detects subgrouping nodes — nodes that specify one plan to each sub group. These nodes are replaced by `AND` nodes (line 5). Line 6 detects agent-plans nodes. This nodes define a set of plans for a specific subgroup. Each of these nodes is replaced by s-combs, in which rows that match the agents in the subgroup, have '1' in the columns that match the allowed states according to the plan. All other rows are all-ones (line 7). Finally, after all the nodes were replaced, we have a tree that matches the s-comb rules representation. The algorithm returns that tree (line 10).

Now, let us demonstrate the algorithm using Figures 17 and 18. The root is a node which its offspring 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 offspring 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 offspring, `SimpleAdvance` and `FlankAttack`. Hence, all of those nodes

Algorithm 3 PlansToRule(plans-tree T).

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

1: **for all** node i in T **do**
2: **if** i's offspring refer to the same group as i **then**
3: replace i with an `OR` node
4: **else if** i's offspring refer to different subgroups **then**
5: replace i with an `AND` node
6: **else if** i's offspring are *agent-plans* **then**
7: replace i with an `s-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
8: **end if**
9: **end for**
10: return T

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 s-comb. For example, the s-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 s-comb includes 'all ones' rows. The rule tree is presented in Figure 18.

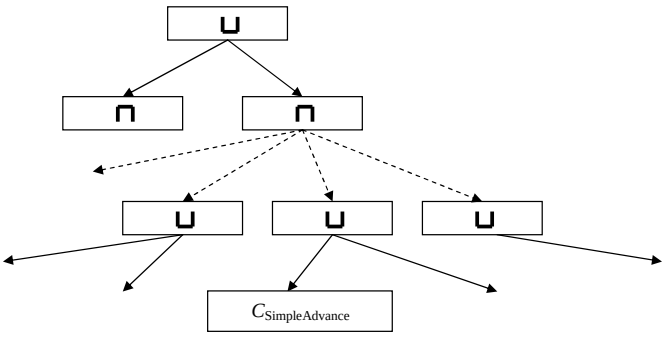

Figure 18: Rule tree.

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. In this section we will examine two possible extensions to the system, that relate to that aspect. These are:

Temporal rules: Adding rules that restrict the set of states an agent might choose according to its current state.

Dynamic policies: Allowing the policy to be changed according to environmental conditions.

The next sections explain these topics in details.

7.1 Temporal Rules

A temporal rule is a rule that further restrict the latitude of the agents' states, by defining allowed 'transitions' from one state to another. That means, that the set of states to which an agent might position itself at time $t+1$ depends on its position at time t. The transition, thus, is a function that maps each state to a set of states, or, in other words, to a superposition:

Definition 7.1 (Transition function). *Let $\mathbb{S} = \langle A, S, B, \lambda, \varphi \rangle$ be some system. A* transition *over E is a function*

$$\tau : S \to \|S\|.$$

The transition lets the agent latitude of choosing certain states to move to (unless consists of one state exactly). However, it clearly defines which states might be chosen and which might not. For example, in the shop system, we may define such rules as:

- An agent may SELL only after NEGOTIATE.
- An IDLE state will not take place after a BREAK.

In fact, each such rule can be broken down to the most fundamental rules: Either can state s_j be chosen after s_i (that is, $s_j \in \tau(s_i)$) or it cannot ($s_j \notin \tau(s_i)$).

We may define as many as m^2 Boolean rules of this kind; from each state to each state. Algebraically, we represent the transition function using a *transition matrix*:

$$M^{8\times 8} = \begin{array}{c} \\ \textsc{Break} \\ \textsc{Idle} \\ \textsc{Negotiate} \\ \textsc{Sell} \\ \textsc{InnerTalk} \\ \textsc{Watch} \\ \textsc{Guard} \\ \textsc{Equip} \end{array} \begin{array}{c} \begin{array}{cccccccc} \textsc{Break} & \textsc{Idle} & \textsc{Negotiate} & \textsc{Sell} & \textsc{InnerTalk} & \textsc{Watch} & \textsc{Guard} & \textsc{Equip} \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Figure 19: State transition matrix.

Definition 7.2 (Transition matrix). *Let $\mathbb{S} = \langle A, S, B, \lambda, \varphi \rangle$ be some system with a transition function τ defined on it. The* transition matrix *is a Boolean matrix M of order $m \times m$ (where $m = |S|$), such that*

$$m_{ij} = 1 \iff s_j \in \tau(s_i).$$

In other words, if m_{ij} is 1, it means that state s_j can follow s_i. If it is 0, then s_j can never be chosen after s_i.

Thus, each row i in this matrix actually represents $\tau(i)$. An example to such a *transition matrix* is given in Figure 19. This matrix, in the shop system, includes the two rules presented above, as well as few others. For instance, in this matrix, $m_{1,2} = 0$, which means that s_2 (IDLE) can never be taken after s_1 (BREAK).

Having the transition matrix, we can predict the states that an agent might choose. If we know that an agent is currently positioned at state s_i, then its next position must be in $\tau(s_i)$. We do not always know the exact position of the agent. Instead, we assume it is superpositioned in one of few possible states. Thus, if the agent is superpositioned in either s_i or s_j, then its next position must be either in $\tau(s_i)$ or in $\tau(s_j)$. In other words, it must be in $\tau(s_i) \cup \tau(s_j)$. In the general case, if, at time t, the agent is superpositioned in some set of states $S_t \subseteq S$, then its position in time $t + 1$ must be in

$$\bigcup_{s_k \in S_t} \tau(s_k).$$

Using the transition matrix, we can easily calculate this algebraically. The superposition of an agent at time t might be given as a Boolean vector $\vec{v}^t$ of order m,

(where m is the number of states in S) in which v_i represents whether s_i belongs to the superposition ($v_i^t = 1$) or not ($v_i^t = 0$). By multiplying it in M, we get a Boolean vector of order m that represents the set of states in which the agent might be superpostioned at $t + 1$:

$$\vec{v}^t \cdot M = \vec{v}^{t+1}.$$

To understand this equation, recall the way we defined the Boolean matrix product (see Definition 3.2). According to this, the ith element in $\vec{v}^{t+1}$ is given by

$$\bigvee_{k=1}^{m} v_k^t \wedge m_{ki}.$$

The meaning of this is that for the element to be 1, it is enough that there is at least one such k that provides $v_k^t \wedge m_{ki} = 1$. m_{ki} is 1 iff transition from state s_k to s_i is allowed. So if the agent might currently be in s_k (i.e., $v_k = 1$) and transition from this state to state s_i is allowed (i.e., $m_{ki} = 1$), the agent might be in s_i at $t + 1$ ($v_i^{t+1} = 1$). Extending this calculation to the whole system is straight forward. This provide a prediction of the next possible states:

Definition 7.3 (Prediction supercoordination). *Let M be a transition matrix over some system $\mathbb{S}$. Let Ω^t be an s-comb which is the interpretation of the agents' observation at time t. The* prediction supercoordination *for the possible states of the agents at time $t + 1$ is*

$$P^{t+1} = \Omega^t \cdot M.$$

The prediction P^{t+1} is an s-coord that describes all the possible states of the agents at time $t + 1$.

Let us now demonstrate a transition matrix in a context of a full system (the shop domain). Suppose that at time t, an agent a_k was superpositioned in states BREAK and IDLE (s_1 and s_2, accordingly). That means, we know — according to observation — that it must be positioned in one of those states. The appropriate row in the s-comb Ω^t will therefore be

	BREAK	IDLE	NEGOTIATE	SELL	INNERTALK	WATCH	GUARD	EQUIP
$\Omega_k^t =$	1	1	0	0	0	0	0	0

Using the transition matrix, we can predict the states to which a_k might move. Intuitively, based on the matrix in Figure 19, we can see the following:

- From state s_1 (BREAK), transition is allowed to all states but s_2, s_4 and s_5 (IDLE, SELL and INNERTALK).
- From state s_2 (IDLE), transition is allowed to all states but s_4 and s_5 (SELL and INNERTALK).

Since we do not know if the agent is currently in s_1 or s_2, we assume that it might move to any of the states in the union of the above options. In this case, the union is all the states but s_4 and s_5. Mathematically, this is exactly what we will get in the kth row of the 'prediction' product $P^{t+1} = \Omega^t \cdot M$:

$$P_k^{t+1} = \begin{array}{c} \text{BREAK} \quad \text{IDLE} \quad \text{NEGOTIATE} \quad \text{SELL} \quad \text{INNERTALK} \quad \text{WATCH} \quad \text{GUARD} \quad \text{EQUIP} \\ (\quad 1 \quad\quad 1 \quad\quad\quad 1 \quad\quad\quad 0 \quad\quad\quad 0 \quad\quad\quad 1 \quad\quad\quad 1 \quad\quad\quad 1 \quad). \end{array}$$

At time $t + 1$, we will have a new observation, which, after interpretation, will produce Ω^{t+1}. Suppose that agent a_k was observed in TALK. In this case, it is interpreted as being superpositioned in {BREAK, NEGOTIATE, INNERTALK, WATCH} — s_1, s_3, s_5 and s_6, accordingly (the interpretation matrix is given in Figure 4). The kth row in Ω^{t+1} is therefore

$$\Omega_k^{t+1} = \begin{array}{c} \text{BREAK} \quad \text{IDLE} \quad \text{NEGOTIATE} \quad \text{SELL} \quad \text{INNERTALK} \quad \text{WATCH} \quad \text{GUARD} \quad \text{EQUIP} \\ (\quad 1 \quad\quad 0 \quad\quad\quad 1 \quad\quad\quad 0 \quad\quad\quad 1 \quad\quad\quad 1 \quad\quad\quad 0 \quad\quad\quad 0 \quad). \end{array}$$

Now, we have two different indications as for the possible superposition of a_k at $t + 1$ — the observation and the prediction. Both must exist. That means, the agnet is superpositioned only in states that are found in *both* Ω^{t+1} and P^{t+1}. This is the *resolution* of the possible positions of the agents at $t + 1$. Formally:

Definition 7.4 (Resolution supercoordination). *Let $\mathbb{S}$ be some environment, with a transition matrix M defined on it. Let P^t be a prediction s-coord for time t (based on observations at $t - 1$), and let Ω^t be an observation s-coord of the agents' states at time t. The* resolution s-coord *of the agents at time t is given by*

$$R^t = \Omega^t \wedge P^t.$$

Thus, an element $r_{ij}^t = 1$ means that at time t, agent a_i might be positioned in s_j, accoording to its action *and* according to prediction from its former action at time $t - 1$. Otherwise, r_{ij}^t will be 0. In our example, the kth row of R^{t+1} will be

$$R_k^{t+1} = \begin{array}{c} \text{BREAK} \quad \text{IDLE} \quad \text{NEGOTIATE} \quad \text{SELL} \quad \text{INNERTALK} \quad \text{WATCH} \quad \text{GUARD} \quad \text{EQUIP} \\ (\quad 1 \quad\quad 0 \quad\quad\quad 1 \quad\quad\quad 0 \quad\quad\quad 0 \quad\quad\quad 1 \quad\quad\quad 0 \quad\quad\quad 0 \quad). \end{array}$$

In this case, the transition matrix prediction helps us narrowing the possible superposition to three possible state. If we had used the observation alone, we would have got four.

Using the above mechanism, we can go one step ahead. Not only can we predict the next states, but we can also better understand the former states. This is important, since the coordination of the agents at time t relies on their coordination at $t-1$. If we find that at $t-1$ there was a fault, it might imply that the current coordination is faulty as well. Remember that using observations does not provide the exact states of the agents, but only a set of possible hypotheses. Therefore, it is possible that at time $t-1$ there was a fault that we could not detect, according to the *optimistic approach* (Section 4).

For that reason, it is important that when we get new information, we will re-examine former states. This can help us detecting faults that had occurred a short time ago. This is important, since this might imply that the cooperation of the agent is broken, and their current states violate the policy. Re-calculation of the states in $t-1$ is done in a very similar way to predicting the states in $t+1$.

By transposing M, we get M^T. This matrix is the opposite of M, in the sense that it describes, for each state, what possible states might have *preceded* it. In M, if m_{ij} is 1, it means that s_j might follow s_i; in this case, $m^T_{ji} = 1$, which means that if at some time t an agent is found in state s_j, then it is possible that its former state was s_j.

Therefore, similarly to the way we have found P^t according to Ω^{t-1}, we can use the resolution s-coord R^t to find the possible states in $t-1$. This reevaluation of the states in $t-1$ is given in the following definition:

Definition 7.5 (Former supercoordination). *Let $\mathbb{S}$ be some environment, with a transition matrix M defined on it. Let R^t be the resolution s-coord for time t (based on the observation Ω^t and the prediction P^t). The* former s-coord F^{t-1} *is the s-coord*

$$F^{t-1} = R^t \cdot M^T.$$

In this s-coord, if an element f^{t-1}_{ij} is 1, it means that the states of agent a_i at time t imply that it might has been in state j at time $t-1$.

Now, we have two indications for the possible states of the agents at $t-1$. First, there is the observation at this time, which is the s-coord Ω^{t-1}. Second, there is the former-supercoordination, F^{t-1}. In fact, in most cases, instead of Ω^{t-1}, we can use the resolution s-coord R^{t-1}, which is based on the predictions from time $t-2$. We can then calculate a better hypothesis of the states at $t-1$:

Definition 7.6 (Bidirectionally evaluated hypothesis). *Let $\mathbb{S}$ be some system. Let R^{t-1} be a resolution s-coord of time $t-1$ (based on observation at $t-1$ and prediction from the states at $t-2$). Let F^{t-1} be a former s-coord of time $t-1$ (based on the resolution s-coord of time t). The* bidirectionally evaluated hypothesis *H^{t-1} is given by*

$$H^{t-1} = F^{t-1} \wedge R^{t-1}.$$

If we replace F^{t-1} and R^{t-1} by their full definition, we will get the following expression:

$$H^{t-1} = (R^t \cdot M^T) \wedge (\Omega^{t-1} \wedge (\Omega^{t-2} \cdot M)).$$

This is why we call it a bidirectional evaluation: For the evaluating of the hypothesis at time $t-1$, it combines former information (from $t-2$) and later information (from t). This helps us find faults that could not be detected immediately. In the same manner, the coordination of former times, such as $t-2$, $t-3$ and so on can be checked as well.

7.2 Dynamic Policy

In the system definition (Definition 3.18), the policy is defined as a Boolean function that maps each coordination to 'legal' or 'illegal'. However, we can extend this to a wider range of system designs. A designer might need a system that defines different policies for different environmental conditions. Mathematically, such conditions of the physical environment provide a set of key/value pairs, as described in the following definition:

Definition 7.7 (Physical environment description). *A physical environment description is a set of key/value pairs, of the form*

$$V = \left\{ \langle \mathit{key}_1, \mathit{value}_1 \rangle, \langle \mathit{key}_2, \mathit{value}_2 \rangle, \ldots, \langle \mathit{key}_g, \mathit{value}_g \rangle \right\}.$$

It is up to the designer to define the environmental keys and their possible values in a specific system. For example, a typical environment description of a military unit that is moving around might be

$$V = \{\langle \text{time}, 17{:}34 \rangle, \langle \text{location}, \text{NorthEast} \rangle, \langle \text{surface}, \text{desert} \rangle\}.$$

In order to have different policies for different environments, the former definition of the policy must be changed. In Definition 3.10, the policy is defined as

$$\varphi : \{\text{coordinations over } E\} \to \mathbb{B}.$$

Now, we should redefine it, as a function that takes two arguments:

Definition 7.8 (Environmental-aware policy). *Let E^C be the set of all possible coordinations over some environment $E = \langle A, S \rangle$, and let V_{all} be the set of all possible physical-environmental descriptions for a given system. The* environmental-aware policy *is a function φ, defined as:*

$$\varphi : E^C, V_{all} \rightarrow \mathbb{B}.$$

For a given coordination $C \in E^C$, and a given description $V \in V_{all}$, the function $\varphi(C, V)$ is 1 iff the coordination C is allowed under the conditions of V.

In Section 5.1, we define the policy using a rule. In order to let the policy be aware of different environmental descriptions, this rule is not enough. A possible solution is to define different rules for different environmental-descriptions. When trying to find out if a specific coordination is legal or not, the policy will determine the rule that satisfies the provided environmental description. However, in this approach, we will have to hold as many rules as the possible environmental descriptions. If few different descriptions require few different policies, then each of them will use a different rule. If those rules are similar in most parts, then a lot of duplicated information will be used. This approach suffers of few major disadvantages. It requires more space to hold it; it requires more time to define it; and it is more susceptible to mistakes and inadequate maintenance.

We suggest a different approach for the definition of such an environmental-aware policy, which we call *dynamic policy*. We take advantage of the rules presentation, which is modular by nature. For this, we introduce a new operator: ' $\rightarrowtriangle$ ' — the conditional operator — in additional to the 'or' ($\sqcup$) and 'and' ($\sqcap$) operators. Like any conditional operator, this operator consists of two parts, the condition and the rule:

$$[\texttt{condition}] \rightarrowtriangle \mathcal{R}_{\text{condition}}.$$

If the logical value of the condition is `TRUE` (i.e. Boolean '1'), then the rule is calculated. Otherwise, it is ignored. Formally, expressions of the type $\mathcal{R} \sqcup ([\texttt{cond}] \rightarrowtriangle \mathcal{R}_{\text{cond}})$ or $\mathcal{R} \sqcap ([\texttt{cond}] \rightarrowtriangle \mathcal{R}_{\text{cond}})$, will be calculated as $\mathcal{R} \sqcup \mathcal{R}_{\text{cond}}$ or $\mathcal{R} \sqcap \mathcal{R}_{\text{cond}}$, accordingly, if `cond = TRUE`. Otherwise, the expressions will be calculated as just $\mathcal{R}$.

Let us demonstrate that by an example from the shop domain. In addition to the common policy of this system, we would like to define a slightly different policy for the Boxing Day. In this day, the storekeeper and the guard are allowed to sell, as long as one of them is guarding. From any other aspect, the Boxing

Day policy is identical to the common one. For this purpose, we define a simple environmental description for this domain, which contains the date. Any check of the legality of a coordination will now be done according to the date. In this example, the policy for the Boxing Day allows all the common coordinations, *plus* those of the 'Boxing Day Rule'. Therefore, we will use the OR operator ($\sqcup$), which is equivalent to coordinations union. Let us notate the common policy with $\mathcal{R}_{\text{common}}$, and the rule that allows the storekeeper and the guard to sell as $\mathcal{R}_{\text{boxing}}$. Each of them is a rule that combines several s-combs with different $\sqcup$ and $\sqcap$ operators. The complete policy rule is

$$\mathcal{R}_{\text{complete}} = \mathcal{R}_{\text{common}} \sqcup ([\texttt{Date = BoxingDay}] \rightarrowtriangle \mathcal{R}_{\text{boxing}}).$$

For example, assume that $\mathcal{R}_{\text{common}}$ is $(C_1 \sqcup C_2) \sqcap (C_3 \sqcup C_4)$, and $\mathcal{R}_{\text{boxing}}$ is $C_5 \sqcup C_6 \sqcup C_7$ (where all the C_k are s-combs). Then the definition of the complete policy is

$$\mathcal{R}_{\text{complete}} = ((C_1 \sqcup C_2) \sqcap (C_3 \sqcup C_4)) \sqcup ([\texttt{Date = BoxingDay}] \rightarrowtriangle (C_5 \sqcup C_6 \sqcup C_7)).$$

This rule will be calculated differently on Boxing Day than on any other day in the year.

In the tree representation, we will notate the left-hand side of the conditional operator — the 'condition' — as a node (call an '`IF`' node). Its single offspring is the root of the right-hand rule in the condition. An example for such a tree is given in Figure 20. In this figure, the root is an `OR` node. Its right-hand subtree will always be calculated. Its left-hand subtree will only be calculated if the condition 'Today = Boxing Day' is `TRUE`.

The algorithm for calculating a dynamic policy rule is based on the algorithm described in Secion 5.2. In that algorithm, the rule tree is searched in a DFS way, and in the way back from the leaves, each node is being flatten according to its type. However, in oppose to other nodes, the `IF` nodes are being calculated in the *first* part of the DFS, that is, during the search downwards.

When getting to such a node, its condition is being checked. If it is false, then this branch of the tree is being completely ignored, as if it is not there. Otherwise, the offspring of the `IF` node is directly connected to its parent, instead of the `IF`. The DFS then continues as usual.

To summarize, the s-combs rule notation provides a natural way for the definition of a dynamically-changed policy that is aware to the physical environment. The complexity of such a rule in the worst case, is identical to this of a static rule (the worst case assumes that all conditions in the tree are `TRUE`). However,

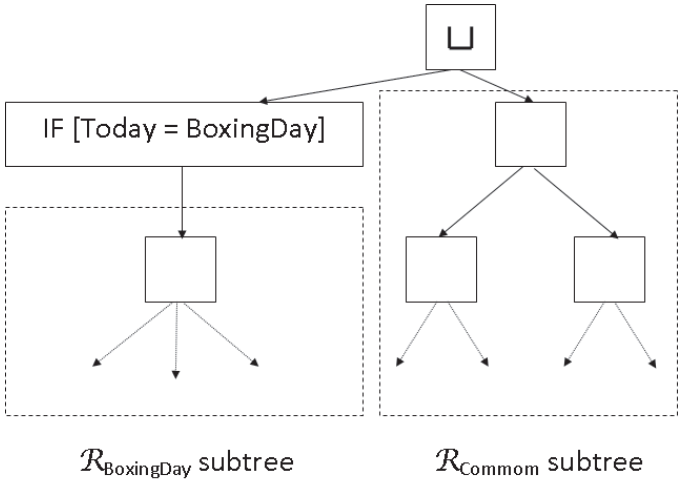

Figure 20: A schematic view of the dynamic ‘shop’ policy

in practice, some of the conditions are false, which leads to a faster calculation. This is a good example for the advantage of this notation, which allows reuse and flexibility.

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 s-combs—which serves as a general framework for coordination design and definition in multi agent systems. The s-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 s-combs are involved and in what kind of relations, and not on the team size.

We presented how the representation of Boolean-matrices rules is modular and flexible. An example for that was introduced, in the reuse of existing systems. Another example is the simplicity of importing other system models, such as hi-

erarchical team model, into the s-comb representation. The last section extends the representation by adding dynamical aspects. The representation advantages are well emphasized in these extensions. Temporal rules are naturally added to the system, utilizing the algebraic properties of matrices. Dynamic policy can be achieved by simply adding conditional operators to the rules.

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, we plan to extend the use of s-comb rules beyond the limit of fault detection. This representation can be used to allow a wide range of utilities, such as proactive fault prevention, decision making, fault analysis and system recovery.

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] Chrysanthos Dellarocas and Mark Klein. An experimental evaluation of domain-independent fault-handling services in open multi-agent systems. In *Proceedings of the Fourth International Conference on Multiagent Systems (ICMAS-00)*, pages 95–102, 2000.

[3] Edmund H. Durfee. Scaling up agent coordination strategies. *IEEE Computer*, 34(7):39–46, July 2001.

[4] Peter Fröhlich, Iara de Almeida Mora, Wolfgang Nejdl, and Michael Schröder. Diagnostic agents for distributed systems. In *ModelAge Workshop*, pages 173–186, 1997.

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

[6] 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.

[7] 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.

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

[9] Meir Kalech and Gal A. Kaminka. Diagnosing a team of agents: Scaling-up. In *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-05)*, 2005.

[10] 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.

[11] Meir Kalech and Gal A. Kaminka. On the design of coordination diagnosis algorithms for teams of situated agents. *Artificial Intelligence AIJ*, 171(8-9):491–513, 2007.

[12] Gal A. Kaminka and Michael Bowling. Robust teams with many agents. In *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-02)*, pages 729–736, 2002.

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

[14] Phil Kim, Brian C. Williams, and Mark Abramson. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

[15] Mark Klein and Chris Dellarocas. Exception handling in agent systems. In *Proceeding of the Third International Conference on Autonomous Agents*, pages 62–68, May 1999.

[16] Gianfranco Lamperti and Marina Zanella. *Diagnosis of Active Systems*. Kluwer Academic Publishers, 2003.

[17] R. Micalizio, P. Torasso, and G. Torta. On-line monitoring and diagnosis of multi-agent systems: a model based approach. In *Proceeding of European Conference on Artificial Intelligence (ECAI 2004)*, volume 16, pages 848–852, 2004.

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

[19] Y. Pencolé, M.O. Cordier, and L. Rozé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. *IEEE Conference on Decision and Control (CDC'02)*, pages 435–440, December 2002.

[20] 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)*, pages 960–967, 2002.

[21] Nico Roos, Annette ten Teije, André Bos, and Cees Witteveen. Multi-agent diagnosis with spatially distributed knowlege. In *Proceedings of the Belgium-Dutch Conference on Artificial Intelligence (BNAIC-02)*, pages 275–282, 2002.

[22] Nico Roos, Annette ten Teije, and Cees Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. In *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-03)*, pages 655–661, July 2003.

[23] Nico Roos, Annette ten Teije, and Cees Witteveen. Reaching diagnostic agreement in multi-agent diagnosis. In *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-04)*, pages 1254–1255, 2004.

[24] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.

[25] M. Sampath, R. Sengupth, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Automatic Control Systems Technology*, 40(9):1555–1575, 1995.

[26] Paul Scerri, Joseph Andrew Giampapa, and Katia Sycara. Techniques and directions for building very large agent teams. In *2005 International Conference on Integration of Knowledge Intensive Multi-Agent Systems, KIMAS'05: Modeling, Exploration, and Engineering*, pages 79–84, April 2005.

[27] Paul Scerri, Régis Vincent, and Roger Mailler. *Coordination of Large-Scale Multiagent Systems*. Springer, October 2005.

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

[29] 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.

[30] 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.

[31] B.C. Williams, P. Kim, M. Hofbaur, J. How, J. Kennell, J. Loy, R. Ragnoand J. Stedl, and A. Walcott. Model-based reactive programming of cooperative vehicles for mars exploration. June 2001.