Towards a Formal Approach to Overhearing: Algorithms for Conversation Identification

Gery Gutnik
Bar-Ilan University
Computer Science Department
gutnikg@cs.biu.ac.il

Gal Kaminka
Bar-Ilan University
Computer Science Department
galk@cs.biu.ac.il

Abstract

Overhearing is gaining attention as a generic method for cooperative monitoring of distributed, open, multi-agent systems. It involves monitoring the routine conversations of agents–who know they are being overheard–to assist the agents, assess their progress, or suggest advice. While there have been several investigations of applications and methods of overhearing, no formal model of overhearing exists. This paper takes steps towards such a model. It first formalizes a conversation system–the set of conversations in a multi-agent system. It then defines a key step in overhearing–conversation recognition–identifying the conversations that took place within a system, given a set of overheard messages. We provide a skeleton algorithm for conversation recognition, and provide instantiations of it for settings involving no message loss, random message loss, and systematic message loss (such as always losing one side of the conversation). We analyze the complexity of these algorithms, and show that the systematic message loss algorithm, which is unique to overhearing, is significantly more efficient then the random loss algorithm (which is intractable).

1. Introduction

Overhearing is fast gaining attention as a generic method for cooperative monitoring of distributed, open, multi-agent systems. Overhearing involves monitoring the routine conversations of agents–who know they are being overheard–to infer information about the agents. Such information can be used to assist them [7,9], assess their progress [6], or suggest advice [1,2,3].

Overhearing is particularly suited to open distributed multi-agent applications, which often use standardized communication protocols. In such settings, agents' internal structure is not generally known to a monitoring agent, but overhearing does not require such knowledge. Instead, the monitoring agent uses the overheard communications as a basis for inference about the other agents. In our paper, we focus on cooperative overhearing, in which the overheard agents know they are being overheard, and do not in any way intend to disrupt the monitor.

Previous investigations of overhearing have demonstrated a range of overhearing techniques. However, these were only in context of specific applications. Novick and Ward [9] have modeled overhearing by pilots that seek to maintain their own situational awareness. Kaminka et al. [6] have developed a plan-recognition approach to overhearing in order to monitor the state of distributed agent teams. Aielo et al. [1] and Bussetta et al. [2,3] have investigated an architecture that enables overhearing, so that domain experts can provide advice to problem-solving agents when necessary. Legras [7] has examined the use of overhearing for maintaining organizational awareness. All these previous investigations have dealt with overhearing without providing a comprehensive formal model of the general problem.

This paper takes first steps towards formalizing the general overhearing task, and its key challenges. In particular, we present a comprehensive theoretical model that is constructed from three components. The first models the representation of conversation protocols, i.e. inter-agent communication templates used to coordinate a specific system task performance (e.g., FIPA interaction protocols [4]). The second component models a complete conversation system, a set of instantiated conversations that take place in a multi-agent system. Finally, the third component of our model represents the view of an overhearing agent on the corresponding conversation system.

We use this model to formulate a key step in overhearing, called conversation recognition, that deals with identifying the conversation that took place, given a set of overheard messages. This is a preliminary step to obtaining information from overheard conversations. We provide a skeleton algorithm for this task and instantiate it for handling lossless and lossy overhearing. We explore the complexity of these algorithms, and

show that handling general lossy overhearing–overhearing where messages can randomly be lost–is computationally expensive. Surprisingly, however, a specific case of lossy overhearing, called systematic message loss – e.g., always losing one side of the conversation, is significantly more efficient in terms of complexity. Fortunately, systematic message loss is likely to be more frequent in practice.

This paper is organized as follows. The next section provides a brief discussion of previous investigations and background. Section 3 formalizes a conversation system, whereas Section 4 presents conversation recognition algorithms. Section 5 discusses these algorithms in terms of their complexity and Section 6 concludes.

2. Background

The work by Nowick and Ward [9] has been an early use of cooperative overhearing to model interactions between pilots and air-traffic controllers. In this model, pilots maintain mutuality of information with the controller not only by dialogue, but also by listening to the conversations of other pilots. While each pilot and controller act cooperatively, the other pilots are not necessarily collaborating on a joint task. Rather, they use overhearing to maintain their s*ituational awareness* out of their own self-interest. Similarly, Legras [7] uses overhearing as a method that allows agents to maintain organizational knowledge. In this approach, agents broadcast changes in their organizational memberships. Other agents use this information to maintain *organizational awareness*.

In contrast, investigations in [1,2,3] describe collaborative settings in which the overhearing agent may act on overheard messages to assist the communicating agents. The settings they describe involve communicating agents, who are engaged in problem solving. An overhearing agent monitors their conversations, and offers expert assistance if necessary.

Kaminka et al. [6] used plan recognition in overhearing a distributed team of agents, which are collaborating to carry out a specific task. Knowing the plan of this task and its steps, the monitor uses overheard messages as clues for inferring the state of different team-members. The authors presented a scalable probabilistic representation (together with associated algorithms) supporting such inference, and showed that knowledge of the conversations that take place facilitates a significant boost in accuracy.

Despite the inspiration and concrete techniques provided by previous work, general challenges in overhearing were only addressed in the context of specific applications. As a result, a model of overhearing is yet to be presented. In this paper, we address this challenge introducing (in Section 3) a formal approach to overhearing.

Moreover, key assumptions made by previous works are difficult to extract. For instance, the investigations, described above, all make the assumption that the overhearing agent can match intercepted messages to a conversation protocol. Most make the assumption that all messages in a conversation are overheard (i.e. no losses). Yet both assumptions are challenged in real-world settings. This paper seeks to address these assumptions by presenting (in Section 4) conversation recognition algorithms.

3. Modeling a Conversation System

Addressing the general overhearing task, we propose a formal model that is constructed of three components: (i) conversation protocols; (ii) a system of conversations using conversation protocols; and (iii) a view on conversations by an overhearing agent.

To demonstrate the proposed model, we consider the following overheard conversation between two agents bidding on a contract. The first agent sends a call for proposal (*cfp*), on which the second agent replies with a proposal–a *propose* message. Then, the first agent accepts this proposal by sending an *accept-proposal*. Finally, the second agent performs the agreed task and communicates an *inform* message–informing the first agent on the established results.

This conversation implements a portion of the FIPA Contract Net protocol [4]. Generally[1], the same protocol can be overheard differently. After the first agent issues a *cfp*, a second agent can *refuse* it or *propose* to it. Then, its proposal is either accepted or rejected by the first agent–communicating an *accept-proposal* or a *reject-proposal* message. Finally, the second agent notifies the first agent on the results of the performed task sending an *inform* or a *failure* message.

In the following sub-sections, we discuss the various components of our model demonstrating them using the presented protocol and conversation.

3.1. Conversation Protocols

When involved in a conversation, agents normally communicate according to a protocol, which can be captured by well-defined patterns. These patterns, i.e. *conversation protocols*, define a template that conversations must follow to achieve a communications

[1] We describe this pattern as it may appear to the overhearing agent.

goal. Hence, conversation protocols specify an abstract representation of the corresponding conversations.

Conversation protocols are widely used in open multi-agent settings. For instance, FIPA protocols [4] are an example to the continuous effort to standardize the use of conversation protocols in multi-agent community. Though frequently used in agent-oriented settings, conversation protocols can be found in human-oriented environments as well. For example, McElhearn [8] has showed that conversation protocols can be extracted by analyzing e-mail mailing list traffic.

In our model, a conversation protocol is a tuple denoted by (R , Σ , S , s_0 , F , δ). Below, we provide a detailed discussion of the components of this tuple.

***Conversation Roles* (*R*):** A conversation role defines a separate functionality in a conversation. Conversation protocols define valid sequences of messages between various conversation roles. Each role determines agent behavior in a specific conversation. In our model, R denotes the set of conversation roles in a conversation protocol. In the contract-net protocol shown above, two roles can be distinguished: the first agent is the *initiator*, whereas the second agent is the *participant* [4]. Thus, the set R consists of these two conversation roles.

***Communicative Act Types* (Σ):** There are often multiple communicative act types (e.g. in FIPA [4]). Σ denotes the set of all communicative act types used by the given conversation protocol. In our example, this set contains: *cfp*, *refuse*, *propose*, etc.

***Conversation States* (*S*):** A conversation state of an agent marks its state within the protocol (in contrast with its internal state). Here, we must distinguish between *individual* and *joint* states [5].

We explain these terms using the contract-net protocol. We denote the two conversation roles as A and B. For the moment, let us consider A individually. The A role starts in an initial conversation state, denoted as A_1, where *initiator* is ready to send a *cfp* message type. Sending this message, the agent transitions to its second conversation state (A_2) in which it has already sent a *cfp* type message and is now waiting to receive either a *propose* or a *refuse* message type. Receiving it, the agent transitions to one of the A_3 or A_4 conversation states, and so on. Similarly individual conversation states A_1-A_8 and B_1-B_8 can be defined over the A and B roles respectively. In this paper, we do not present a detailed discussion on the protocol implementation, but we refer the readers to [5] for additional information.

The same protocol may also be defined using a collection of joint interaction states [5], $S = S_A \times S_B$, where each member of S corresponds to a specific combination of individual states. However, not all joint states are legal. For example, A_1B_1 is a legal joint conversation state in the given conversation protocol, whereas A_2B_1 joint conversation state is considered to be illegal (since it denotes a state where A has sent a message but B did not receive it). In our model, S is the set of all legal joint conversation states over a conversation protocol. In our example, S contains the following joint conversation states: A_1B_1, A_2B_2, etc.

***Initiating Conversation State* (s_0):** s_0 is an initiating joint conversation state, which corresponds to the combination of the initiating individual conversation states over the various conversation roles.

***Terminating Conversation States* (*F*):** F defines the set of joint conversation states that terminate the conversation. Thus, $F \subseteq S$. In our example, F includes A_3B_3 joint conversation state in which the *initiator* received a *refuse* message and terminated, whereas the *participant* has sent it and terminated as well.

***Transition Function* (δ):** δ determines the progress of a conversation by defining which message types are expected at different points of the conversation according to its current conversation state.

In order to define δ, we must first define following parameters. An abstract message am is a $<r_x,r_y,\sigma>$, which is a member of the relation AM, where $AM=\{<r_x,r_y,\sigma> \mid r_x, r_y \in R, \sigma \in \Sigma$ and $r_x \neq r_y\}$. Thus, AM denotes a set of abstract messages that may potentially correspond to the appropriate conversation protocol.

Now, we define δ as $S \times AM \rightarrow S$. Thus, the δ function defines whether a transition, between two legal joint conversation states, is possible. In addition, δ determines the specific abstract message, which causes this transition to occur.

In the example above, let us consider the $\delta(A_1B_1,<A,B,cfp>) = A_2B_2$ instance of δ. This instance has the following interpretation: given agents in A_1B_1 joint conversation state, the $<A,B,cfp>$ abstract message (of a *cfp* message type sent from the *initiator* to the *participant*) causes the agents to transition to the A_2B_2 joint conversation state.

Based on the presented definition of conversation protocols, we can now define the set of possible abstract conversation sequences over a conversation protocol. Given a conversation protocol p, we denote this set as $AS(p)$. To define this parameter, we define a transition function on a sequence of abstract messages. This function, defined as $\delta^*: S \times AM^* \rightarrow S$ (where AM^* denotes

the set of all possible sequences over *AM*), can be formulated recursively as follows:

$$\delta^*(s,\varepsilon) = s$$

$$\delta^*(s, wv) = \delta(\delta^*(s,w), v)$$

where $s \in S$, $\varepsilon(empty) \in AM^*$, $v \in AM$, $w \in AM^*$

Using δ^*, we define the $AS(p)$ set. An abstract conversation sequence is considered to be possible over a given conversation protocol if and only if it is a sequence of abstract messages that begins from an initiating conversation state and ends in one of the terminating conversation states. Thus, given a conversation protocol p denoted by a tuple (R , Σ , S , s_0 , F , δ), the $AS(p) \subseteq AM^*$ is defined as:

$$AS(p) = \{ w \in AM^* \mid \delta^*(s_0, w) \in F \}$$

In the presented FIPA protocol, there are four possible abstract conversation sequences [5]. Let us consider one of them: <*A,B,cfp*> <*B,A,propose*> <*A,B,accept-proposal*> <*B,A,inform*>. This sequence corresponds to $s_0 = A_1B_1 \rightarrow \ldots \rightarrow A_8B_8 \in F$ sequence of joint conversation states. In fact, this abstract conversation sequence corresponds to the conversation described at the beginning of Section 3.

3.2. Conversation Systems

A conversation system is a set of conversations in a multi-agent system. In our model, a conversation system is denoted by a tuple (P , A , Λ , I , C). In this section, we describe these components in details.

***Conversation Protocols* (*P*):** P is the set of conversation protocols of the conversation system, where each protocol is defined by a tuple as shown in Section 3.1.

***Agents* (*A*):** A indicates the set of agents in the corresponding conversation system. Based on this parameter, we define another element in the model–2^A. Using its formal definition–2^A is the set of all subsets of A–we refer to it as the set of all possible conversation groups in the conversation system. However, following the intuition that at least two agents must be involved in a conversation, we further restrict the definition of the 2^A set to be formulated as $2^A = \{ g \mid g \subseteq A \text{ and } |g| \geq 2 \}$.

***Conversation Topics* (Λ):** Λ denotes the set of all conversation topics in the conversation system.

***Intervals* (*I*):** An interval is a time period within the conversation system lifetime. Thus, we define I as follows: $I = \{ [t_1 , t_2] \mid t_1, t_2 \text{ time stamps }, t_1 \geq 0 , t_2 \leq lifetime , t_1 \leq t_2 \}$.

***Conversations* (*C*):** A conversation in a conversation system is defined by a group of agents $g \in 2^A$ implementing a conversation protocol $p \in P$ on a conversation topic $\lambda \in \Lambda$ within a time interval $i \in I$ using an abstract conversation sequence $am^* \in AS(p)$. We can formulate the set of conversations, which is denoted as C, in a conversation system as follows:

$$C \subseteq \{ (p , g , \lambda , i , m^*) \mid p \in P , g \in 2^A , \lambda \in \Lambda ,$$
$$i \in I , m^* \xleftarrow{\mathbf{g},\boldsymbol{\lambda},i} am^* \in AS(p) \ \}$$

Thus, a conversation $c \in C$ in a conversation system is a tuple (p , g , λ , i , m^*). Here, the m^* parameter of conversation needs further explaining. This parameter denotes the actual conversation sequence that has taken place in the corresponding conversation. In fact, m^* is an implementation of some abstract conversation sequence over the corresponding conversation protocol. The actual conversation sequence m^* instantiates an abstract conversation sequence $am^* \in AS(p)$ with conversation group g, topic λ and time interval i. This instantiation is established as follows:

[1] Instantiating conversation roles with agents: Here, we determine the mapping between conversation roles of the conversation protocol and the agent conversation group implementing it. For every conversation role $r \in R$, we determine an agent $a \in g$ ($g \in 2^A$) implementing it.

[2] Instantiating abstract messages with a topic: Each abstract message of the implemented abstract conversation sequence is instantiated with the same topic, i.e. the topic of the conversation.

[3] Instantiating abstract messages with time stamps: Finally, each abstract message of the implemented abstract conversation sequence is instantiated with a time stamp within a given time interval.

A conversation sequence m^* can therefore be denoted as $m^* = \mu_1 \ldots \mu_n$ where μ_i ($\forall i$, $i=1,..,n$) denotes a message within the conversation sequence. A single message is defined as $\mu = <s,r,\sigma,\lambda,t>$, where s and r are the sender and the recipient of the message (s, $r \in g$), σ is its message type, λ is its topic and t is its time stamp.

To demonstrate this formalization, we return to the conversation described at the beginning of Section 3. We denote the two conversing agents as $agent_x$ and $agent_y$, and their conversation group as $g = \{agent_x, agent_y\}$. Accordingly, the first agent is the *initiator*, while the second is the *participant*. We denote the topic of this conversation as $\lambda = contract\text{-}x \in \Lambda$. Finally, we denote the interval of this conversation as $i = [t_1, t_4]$ assuming that the messages have been communicated at t_1, t_3, t_3, and t_4 time stamps. Thus, the actual

conversation sequence of the given conversation can be represented as <$agent_x$, $agent_y$, *cfp, contract-X,* t_1><$agent_y$, $agent_x$, *propose, contract-X,* t_2><$agent_x$, $agent_y$, *accept-proposal, contract-X,* t_3><$agent_y$, $agent_x$, *inform, contract-X,* t_4>.

3.3. Overhearing Conversations

An overhearing agent monitors inter-agent conversations by listening in to the exchanged communications. We denote the observed conversation sequence as o^* as opposed to m^*. The actual conversation sequence m^* is defined as $m^*=\mu_1 \ldots \mu_n$ where μ_i ($\forall i$, $i=1,..,n$) denotes a message within the actual sequence. Analogously, we define the observed conversation sequence as $o^*=o_1 \ldots o_m$ in which o_i ($\forall i$, $i=1,..,m$) denotes an observed message of o^*.

Since the overhearing agent may not overhear all messages, or may incorrectly overhear some messages, the overheard conversation sequence does not necessarily match the actual conversation sequence. Table 1 summarizes the possible differences between the two conversation sequences.

	Loss (m<n)	**Insert** (m>n)	**Order** (m=n)
Sequence Level	Losing some messages of the actual sequence.	Misoverhearing the actual sequence or misclassifying messages of another sequence.	Inaccurately overhearing the order of messages in actual sequence.
Message Level	**Errors and Losses** ($o_i \neq \mu_j$)		
	Missoverhearing or losing some information of the overheard message (e.g. can not resolve the designated recipient of overheard message).		

Table 1. Possible differences between actual and overheard conversation sequences.

4. Conversation Recognition Algorithms

Overhearing a conversation sequence o^*, one of the key objectives of the overhearing agent is to correctly recognize its appropriate conversation within the conversation system. Specifically, the agent should determine its conversation group (g), topic (λ), and interval (i). It must also identify the appropriate protocol (p) and its actual conversation sequence (m^*). We focus on the extraction of p and m^*, since extracting the other elements is almost trivial in many practical settings.

We propose a skeleton algorithm to determine the protocol corresponding to an observed sequence of messages o^* (Figure 1). Finding a matching protocol also enables us to determine its m^*.

The proposed skeleton algorithm follows similar principles to the debugging algorithm applied in [10]. The algorithm consists of three phases. Phase I is initialization (lines 1-2). Here, we construct a potential protocol set (*PP*) over *P*, which assumed to be given in advance. Each protocol in *PP*, called a *control protocol*, is an extension of the original protocol including a control mechanism used for performing phases II-III of the algorithm. At phase II (lines 3-12), we disqualify inappropriate protocols. For each observed message, each potential protocol is checked (line 8) using *CheckObsMsgMatch*. Inappropriate protocols are accumulated in the disqualified protocol set (*DP*) (line 9) and are subtracted from the *PP* set at the end of each iteration (line 11). Finally, at phase III (lines 13-14), we determine the final protocols, out of whatever protocols remain in the set *PP*.

```
Algorithm FindMatchingProtocols
  (input : observed sequence o* := o1 o2 ... om
   output : protocol set ⊆ P)
 1: // Phase I - Initialiaze
 2: PP = InitializePotentialProtocols(P)
 3: // Phase II - Disqualify inappropriate protocols
 4: foreach oi in o*
 5:    if PP is empty then break
 6:    DP = empty set
 7:    foreach pp in PP
 8:       bool rc = CheckObsMsgMatch( oi , pp)
 9:       if not rc then DP = DP ∪ {pp}
10:    end foreach
11:    PP = PP \ DP
12: end foreach
13: // Phase III - Determine final protocols
14: return DetermineFinalProtocols(PP)
```

Figure 1. FindMatchingProtocols algorithm

This algorithm is a generic skeleton. Different instantiations are needed to handle the problems described in Table 1. Below, we first show an overhearing algorithm for lossless o^* (Section 4.1). We remove this naïve assumption, first in general lossy overhearing (Section 4.2), and then in systematic lossy overhearing (Section 4.3).

4.1. The Naïve Algorithm

The Naïve algorithm assumes that the observed conversation sequence is equal to the actual conversation sequence, i.e. it assumes no losses.

In this case, *InitializePotentialProtocols* extends the original conversation protocols with two new components. The first is $s_{curr} \in S$ – a pointer to the current conversation state within the protocol–it is initialized to s_0. The second is *AG* – a mapping between *R* and *A*– whose elements are initialized to *unknown*. We use the

AG mapping to accumulate information about agents implementing various roles of the protocol.

Then, we check (*CheckObsMsgMatch*) whether exists a transition from s_{curr} to some s_{next} that is appropriate to the communicative act type of *o*. We also check whether agents, corresponding to this message, match the information in *AG* (*CheckRolesMatch*). In case these two conditions are satisfied, s_{curr} is incremented to s_{next} and procedure returns *true*, else it returns *false*.

Finally, each protocol, remaining in *PP*, is checked (*DetermineFinalProtocols*) to determine whether its $s_{curr} \in F$. If so, the corresponding protocol is considered as matching the observed conversation sequence.

4.2. The Random Loss Algorithm

The Random Loss algorithm handles the case in which there are multiple random message losses in *o**, where each such loss is made up to *k* consecutive messages. This lossy overhearing condition may occur, for example, in case of malfunction in the overhearing agent, due to which it loses a certain interval within the overheard conversation.

In our example, in case *k=2*, this algorithm can determine that o^*=<$agent_x$, $agent_y$, *cfp, contract-X,* t_1> <$agent_y$, $agent_x$, *inform, contract-X,* t_4> corresponds to the FIPA protocol introduced in Section 3. Furthermore, keeping track of the conversation state sequence within the protocol, it may be able to restore *m**.

In the Random Loss algorithm, control protocols are initialized with two additional components–*CS* and *AG* (*InitializePotentialProtocols*). The *AG* mapping has identical semantics as before. However, instead of a single s_{curr}, the *CS* set contains numerous pointers to the possible current conversation states reflecting the uncertainty caused by losing messages.

In *CheckObsMsgMatch* (Figure 2), for each s_{curr} in *CS* (lines 2-4), we determine its possible next states using *PropIgnLostMsg*. These next possible states are accumulated in *NS* set (line 3), which is then assigned to *CS* (line 5). If at the end of the procedure, *CS* is not empty, the procedure returns *true*, else it returns *false*.

Given a s_{curr} state, *PropIgnLostMsg* (Figure 3) determines its next possible states ignoring up to *k* consecutive losses. In each iteration, we apply two sets – NS^i and IS^{i+1}. The first contains the next possible states corresponding to iteration *i* (line 11), whereas the second set holds up the intermediate states that are to be checked in the following iteration *i+1* (line 12).

Finally, we determine final protocols using procedure similar to the one shown in Figure 3. A protocol is considered to be final if in its *CS* set there is at least one state which is either final or there is a final state with no more than *k* consecutive losses from it.

```
Procedure CheckObsMsgMatch
  (input : observed message o := (sen, rcv, σ, λ, t),
           control protocol pp := (p, CS, AG)
                        where p := (R, Σ, S, s0, F, δ)
   output : bool)
1: NS = empty set
2: foreach s_curr in CS
3:    NS = NS ∪ PropIgnLostMsg (s_curr, o, AG)
4: end foreach
5: CS = NS
6: return not (CS is empty)
```

Figure 2. CheckObsMsgMatch procedure

The Random Loss Algorithm

```
Procedure PropIgnLostMsg
  (input : conversation state s_curr,
           observed message o := (sen, rcv, σ, λ, t),
           agent-role mapping AG
   output: conversation state set NS)
1: NS = empty set
2: IS^0 = { s_curr }
3: for i = 0 to k
4:    if IS^i is empty then break
5:    NS^i = IS^(i+1) = empty set
6:    foreach s_int in IS^i
7:       bool exists = check whether exists
8:                     δ(s_int, <r_x, r_y, σ>) = s_next
9:       if exists and
10:         CheckRoles Match (o, <r_x, r_y, σ>, AG)
11:         NS^i = NS^i ∪ { s_next }
12:      IS^(i+1) = IS^(i+1) ∪ { s | δ(s_int, _) = s }
13:   end foreach
14: end for
15: NS = ∪_{i=0}^{k} NS^i
16: return NS
```

Figure 3. PropIgnLostMsg procedure

4.3. The Systematic Loss Algorithm

The Systematic Loss algorithm handles a more common situation in lossy overhearing—losing up to *l* conversation roles. This condition can occur in case that an overhearing agent, due to its location, cannot overhear messages sent from agents implementing the lost roles (e.g., the overhearing agent sees outgoing messages, but not incoming messages). In our example, in case *l=1* and the lost role is *initiator*, the algorithm can determine that o^*=<$agent_y$, $agent_x$, *propose, contract-X,* t_2><$agent_y$, $agent_x$, *inform, contract-X,* t_4> corresponds to the FIPA protocol described in Section 3.

In the Systematic Loss algorithm, we determine for each set of lost roles (*LR*) a *CS* and *AG* component. Thus, for each potential protocol, we define a control set (*CLR*) that contains (*LR*,*CS*,*AG*) tuples.

```
Procedure CheckObsMsgMatch
  (input : observed message o:=(sen,rcv,σ,λ,t),
           control protocol pp:=(p,CLR)
           where p:=(R,Σ,S,s0,F,δ) and
                 CLR={(LR,CS,AG)|∀LR∈LRS}
   output bool)
 1: foreach (LR,CS,AG) in CLR
 2:   NS=empty set
 3:   foreach s_curr in CS
 4:     bool exists= check whether exists
 5:                δ(s_int,<r_x,r_y,σ>)=s
 6:     if exists and
 7:       CheckRolesMatch(o,<r_x,r_y,σ>,AG)
 8:         NS=NS ∪ PropIgnLostRoles(s,LR)
 9:   end foreach
10:   CS=NS
11:   if CS is empty then
12:     CLR=CLR \{(LR,CS,AG)}
13: end foreach
14: return not (CLR is empty)
```

Figure 4. CheckObsMsgMatch procedure

The Systematic Loss Algorithm

```
Procedure PropIgnLostRoles
   (input : conversation state s_curr,
            conversation role set LR ⊆ R
    output : conversation state set NS)
 1 : NS = empty set
 2 : IS^i = {s_curr}
 3 : while IS^i is not empty
 4 :   IS^(i+1) = empty set
 5 :   foreach s_int in IS^i
 6 :     foreach δ(s_int,<r,_,_>) = s
 7 :       if r ∈ LR
 8 :         IS^(i+1) = IS^(i+1) ∪ {s}
 9 :       else
10 :         NS = NS ∪ {s_int}
11 :       end if
12 :     end foreach
13 :   end foreach
14 :   IS^i = IS^(i+1)
15 : end while
16 : return NS
```

Figure 5. PropIgnLostRoles procedure

In *CheckObsMsgMatch* (Figure 4), each (*LR*,*CS*,*AG*) is considered individually (line 1). For each s_{curr} in its *CS*, we determine in lines 4-9 whether exists a potential next state and then propagate from it ignoring the lost roles (using *PropIgnLostRoles* in Figure 5). The next potential states are accumulated in *NS* set, which is later assigned to *CS* (line 10). If *CS* is empty, the (*LR*,*CS*,*AG*) tuple is discarded from *CLR* (lines 11-12). The procedure returns *true* if at the end of it the *CLR* set is not empty, else it returns *false* (line 14).

5. Discussion

We now turn to analyzing the complexity of the conversation recognition algorithms we presented. This section analyzes these algorithms in terms of their complexity. The algorithmic skeleton (Figure 1) consists of three phases. However, only phases II and III contribute to algorithm complexity. In phase II, we match each of *m* messages with each protocol in *PP* using *CheckObsMsgMatch*. In phase III (*DetermineFinalProtocols*), we determine whether a protocol is final (*CheckIfProtocolFinal*). In this analysis, we denote the complexity of *CheckObsMsgMatch* at iteration *i* as $O(f_1^i)$, while *CheckIfProtocolFinal* procedure is denoted as $O(f_2)$.

The complexity of both phases depends on the number of protocols in *PP* at each stage of the algorithm. In the best case, all (but one) protocols are disqualified after the first iteration, whereas the final protocol remains through all *m* iterations. Assuming *m* is relatively big, the complexity of disqualifying |*P*|-1 protocols is negligible. Thus, the best-case complexity can be formulated as follows:

$$(|P|-1)\mathbf{O}(f_1^1)+\sum_{i=1}^{m}\mathbf{O}(f_1^i)+\mathbf{O}(f_2)=\sum_{i=1}^{m}\mathbf{O}(f_1^i)+\mathbf{O}(f_2)$$

In the worst case, all protocols remain consistent with all *m* messages, and are therefore repeatedly matched against overheard messages:

$$|P|(\sum_{i=1}^{m}\mathbf{O}(f_1^i)+\mathbf{O}(f_2))$$

Now, let us focus on evaluating the $\Sigma O(f_1^i)+O(f_2)$ component–the complexity for matching a single protocol–for each algorithm. We denote this complexity by O(*T*). In the Naïve algorithm, both $O(f_1^i)$ and $O(f_2)$ are equal to O(1), since both procedures only perform a simple check. Thus, O(*T*)=O(*m*) for the Naïve algorithm.

In the Random Loss algorithm, the complexity of $O(f_1^i)$ and $O(f_2)$ depends on the size of the appropriate *CS* set. The size of *CS* in iteration *i* is determined by the *NS* set of the previous iteration (*i-1*). Furthermore, for each state in *CS*, we examine all states that are up to *k* transitions from it. Thus, in order to evaluate O(T), we must consider the structure of δ function and the size of the *NS* set established in each iteration.

In the best case for O(T), δ contains only one possible transition for each state and |*NS*| is always equal to 1. Accordingly, O(T) is equal to $mO(k)+O(1)=O(mk)$.

In the worst case, δ contains *b* transitions for each state – *b* is the branching factor of the state ($1 \le b \le |\Sigma|$).

Thus, the complexity $O(\alpha)$ of examining up to k transitions from a certain state $s \in S$ can be evaluated as $O(1+b+b^2+\ldots+b^k) \leq O(b^{k+1})$. Such states contribute no more than one new state to *NS*. In the worst case, $|NS|$ is $1+b+b^2+\ldots+ b^{k-1} \leq b^k$. We denote it as β. Thus, the complexity of $\Sigma O(f_1^i)$ is $\alpha \cdot (1+\beta+\beta^2+\ldots+\beta^m) \leq \alpha\beta^{m+1} = O(b^{mk+2k+1})$. Analogously, the complexity of $O(f_2)$ is $b^m(1+b+b^2+\ldots+b^{n-m}) = O(b^{n+1})$, where $n = |m^*|$. Thus, the worst-case complexity of O(T) for the Random Loss algorithm is $O(b^{mk})+O(b^n)$.

In the Systematic Loss algorithm, the complexity of $O(f_1^i)$ and $O(f_2)$ depends on the size of *CS* and the structure of δ. However, this complexity also depends on the number of *LR* sets in *CLR*, i.e. $|CLR|$. In Section 4.3, we have defined each *LR* set as a possible combination of up to l lost roles of the protocols' conversation roles. Thus, $|CLR|$ can be formulated as follows:

$$|CLR| = \sum_{i=0}^{l} \binom{|R|}{i}$$

In the best case, $b=1$ and $|NS|$ is always equal to 1. In addition, all *LR*s (but one) are disqualified after the first iteration. Furthermore, from each state, h states can be skipped (*PropIgnLostRoles*). Thus, the complexity of $O(f_1^i)$ is always $O(1+h)=O(h)$, and the complexity of $O(f_2)$ is O(1)–simply checking the remaining state. Therefore, the best-case complexity of O(T) for the Systematic Loss algorithm is equal to $O(mh)$.

In the worst case, we assume that no propagation can be made. Thus, the complexity of $\Sigma O(f_1^i)$ is similar to the Naïve algorithm, only multiplied by $|CLR|$, i.e. $|CLR|O(m)$. As for $O(f_2)$, from the single state in *CS*, all states in levels n-m from it must be examined. Thus, similarly to the principles explained above, $O(f_2) \leq |CLR|O(b^{n-m+1}) = |CLR|O(b^{n-m})$. Thus, the worst-case complexity of O(T) for the Systematic Loss algorithm is equal to $|CLR|(O(m)+O(b^{n-m}))$.

In general, it is difficult to determine which of the algorithms is better. However, in practice, we often know which roles are lost or at least know the number of lost roles. In such cases, the $|CLR|$ parameter becomes a constant and, thus, the Systematic Loss algorithm seems to be more efficient than the Random Loss algorithm.

6. Conclusions and Future Work

In this paper, we have taken the first steps towards a formal approach to overhearing. Using the proposed theoretical model, we were able to formulate a key problem of overhearing – conversation recognition.

Addressing this problem, we discuss conversation identification, given a set of overheard messages. Here, we present a skeleton algorithm (and three instantiations) for finding a conversation pattern that corresponds to the overheard messages, despite losses of overheard messages. We show that the Naïve algorithm, assuming no losses, is efficient. However, its naïve assumption is challenged in real-world settings.

Addressing lossy overhearing, we analyze the best-case and the worst-case complexities of the Random Loss and the Systematic Loss algorithms. We show that, in general, it is difficult to determine which of the two algorithms is better. However, in practice, we expect the Systematic Loss algorithm to outperform the Random Loss algorithm.

In this paper, we analytically derived the best- and worst-case complexities of the algorithms. In the future, we hope to examine their performance empirically. Furthermore, conversation recognition is but a first step towards a general formal treatment of overhearing. We plan to tackle additional overhearing challenges, as mentioned in the paper.

References

[1] Aielo, M., Busetta, P., Dona, A. & Serafini, L. (2001). Ontological overhearing. In *Proceedings of ATAL-2001*. Seattle, USA.

[2] Busetta, P., Serafini, L., Singh, D. & Zini, F. (2001). Extending multi-agent cooperation by overhearing. In *Proceedings of CoopIS 2001*, Trento, Italy.

[3] Busetta, P., Dona, A. & Nori, M. (2002). Channeled multicast for group communications. *In Proceedings of the AAMAS-02*. Bologna, Italy.

[4] FIPA Specifications (2004). FIPA Specifications, *at www.fipa.org/specifications/index.html*.

[5] Gutnik, G. & Kaminka, G.A. (2004). A comprehensive Petri net representation for multi-agent conversations. MAVERICK Technical Report 2004/1, Department of Computer Science, Bar-Ilan University, *at www.cs.biu.ac.il/~maverick/tech-reports/*.

[6] Kaminka, G.A., Pynadath, D.V. & Tambe, M. (2002). Monitoring teams by overhearing: a multi-agent plan-recognition approach. *JAIR*, 17, 83-135.

[7] Legras, F. (2002). Using overhearing for local group formation. *In Proceedings AAMAS-02*. Bologna, Italy.

[8] McElhearn, K. (1996). *Writing conversation: an analysis of speech events in e-mail mailing lists*. Masters' thesis, Language Studies Unit, Aston University.

[9] Novik, D. G. & Ward, K. (1993). Mutual beliefs of multiple conversants: a computational model of collaboration in air traffic control. In *Proceedings of AAAI-93*, pp. 196-201. Washington, DC, USA.

[10] Poutakidis, D., Padgham, L. & Winikoff, M. (2002). Debugging multi-agent systems using design artifacts: The case of interaction protocols. *In Proceedings of AAMAS-02*, pp. 960-967. Bologna, Italy.