Establishing Trusty Mutual-Belief among Cooperative Agents

Wenpin Jiao
Department of Computer Science
University of Victoria
Victoria, BC, Canada V8W 3P6

Abstract: Believing mutually is one important premise to ensure that cooperation among multiple agents goes smoothly. However, mutual belief among agents is always considered for granted. In this paper, we adapt a method based on the position-exchange principle (PEP) to reason about mutual belief among agents before cooperation starts. By reasoning about mutual belief among agents, we can judge whether cooperation among agents can go on rationally or not.

On the other hand, if there are malicious agents involved in cooperation, even though agents believe mutually while cooperating, the profit of honesty agents will be injured. Especially in the e-commerce environment, it is crucial to establish a fair exchanging circumstance over the Internet, in which malicious agents should be forbidden from transactions. To establish trusty mutual-belief and make cooperation useful, agents should be able to reason about cheating behaviours of malicious agents during cooperation.

In this paper, we put forward a process algebra approach by extending the standard pi-calculus to model cooperation plans and agents. For an agent, we define a new kind of pro-activity, expectations, to describe what action the agent expect to occur and how much the agent is satisfied with the action's occurrence. By extending the standard pi-calculus, we can specify the expectations of agents and reason about the satisfaction degrees of agents during their evolutions.

To reason about mutual belief formally, we bind the position-exchange principle into the inference rules defined on process expressions; whereas to reason about cheating behaviours among agents during cooperation, we define a group of criteria for anti-cheating. In this paper, we also give some examples to show how agents can be formally defined in the process algebra, how agents build mutual-belief by using the PEP inference rules, and how agents reason about cheating by referring to their anti-cheating criteria during commercial transactions.

Keywords: Intelligent Agent, Cooperation, Mutual-Belief, Anti-Cheating, Position-Exchange Principle, Pi-Calculus

1 Introduction

In computer science, there are so many great challenges to build computer systems that can work together [11]. Computer systems are becoming more and more complex, and it is a greater challenge to integrate intelligent autonomous systems (e.g. agents) together. Cooperation among agents is one of the keys to drawing multiple intelligent systems together. Moreover, cooperation is a key concept that differentiates multi-agent system from other related disciplines such as distributed computing, object-oriented systems, and expert systems [4].

Cooperation among multiple agents should meet some specific features or criteria, such as 1) agents should response mutually, 2) all agents should make joint commitments, 3) each agent should be committed to supporting inter-actions [3]. That is to say, every agent participating in cooperation must believe that any other agents are honest and responsible and will take actions following a specific cooperation plan, and vice versa. In other words, all agents involved in a specific cooperation must believe each other mutually.

However, in most of cooperation schemas, such as the FA/C model [12], Joint Intention Framework [13][21], and SharedPlan [6], mutual belief is considered for granted. Thus, if there is a malicious agent who is allocated some cooperative tasks, the cooperation plan will fail eventually. So all agents should build mutual belief before cooperation. By reasoning about mutual belief among agents, we can further judge whether a cooperation plan is feasible and its task allocation is reasonable.

In fact, when an agent involved in cooperation acts, it should comply with a specific behaviour specification consistent with the cooperation plan. Meanwhile, once after the agent takes an action, it must
expect to observe a specific result or response from others so that it could conclude whether it can believe others or whether it itself is believed by others. If any agents participating in cooperation believe that they are trusted by others and the others are also believable, we will be able to say that those agents believe each other mutually and cooperation will proceed smoothly.

However, in a distributed system, an agent almost knows nothing about others and even does not know whether the others have received the message it had sent, so it can only reason about the others’ knowledge based on its own knowledge. To achieve that, an agent has to take for granted that others will think and act in a similar way as itself. In this paper, we adopt a technique using the position-exchange principle to reason about mutual belief between agents.

The position-exchange principle means that one will put him in others’ position and judge others’ feelings by his own. In other words, when one wants to reason about another, he will take the view of the other and thinks as if he were the other. For example, when Alice wants to reason about Bob’s knowledge, she can take the place of Bob and thinks as if she herself were Bob. For instance, “If I did it under this circumstance, I believe that if Bob were me he would do it under the similar circumstance, too.”

In a logic system, the position-exchange principle can be formally described as the following formula.

\[ B_x(\alpha \rightarrow \beta) \rightarrow B_y(B_x(\alpha[B/A] \rightarrow \beta[B/A])) \]

Where, \( B_x \) indicates that \( x \) believes \( y \) is held; \( \alpha[B/A] \) is a new formula different from \( \alpha \), in which all variables related to \( A \) are substituted or exchanged with those corresponding variables related to \( B \).

It means if \( A \) believes that \( \beta \) will be held under condition \( \alpha \), \( A \) will believe as well that \( B \) will believe the similar conclusion \( \beta[B/A] \) will be held under the similar condition \( \alpha[B/A] \).

Since \( A \) can only start from its own view to reason about \( B \)'s belief, it does not know how \( B \) acts. From the above formula, we can find that when we were using the position-exchange principle, we not only substituted those variables related to agents, but also transformed the actions corresponding to agents. However, in a general logic framework, we cannot reason about actions, let alone transforming actions to adapt to the position-exchange principle. So we use process algebra, the pi-calculus, to reason about mutual belief among agents. In the pi-calculus, actions of the pi-calculus processes occur in pairs and are mainly for communicating. Thus when we use the position-exchange principle, we can reason about other’s belief by substituting both variables and mutual complementary input/output actions.

To reason about others’ beliefs based on messages sent or received, an agent must be ensured that the messages it sends must be able to be received by the destination agents and the messages it receives must come from ones whom it is willing to communicate with. We assume that the distributed environment in which agents live is secure, in which messages will reach their destinations eventually and agents can get the messages that they are waiting for from correct senders.

As we have pointed out that an agent must comply with a cooperation plan when it participates in cooperation and takes actions, to specify a concrete agent, we should first know what the plan is and what an agent should do to carry out its tasks. In the following sections, we will first give the formal framework that we use to formalize cooperation plans and agents in section 2.

Then in section 3, we will formally describe what an agent, a cooperation plan, and cooperation look like. In our formalism, a cooperation plan has a hierarchical structure and is defined as a composition of pi-calculus processes, in which nodes are corresponding to tasks to be allocated to agents. To indicate that agents are cooperating, we allow there are some specific relationships among nodes within the hierarchical structured plan, such as synchronization, serialization, parallel, and so on. For an agent, it not only can react to its environment but also possesses some kinds of pro-active attitudes, such as beliefs, desires, and goals [16] [22] [20]. Then an agent can be viewed as an entity that has its own behaviours and behaviour specification, is assigned with specific tasks, and has some beliefs on it and on others. In this paper, we define a new kind of pro-activity for agents, i.e., expectations. For an expectation of an agent, it indicates what action the agent expects to occur subsequently. Thus, a multiple cooperative agent system can be defined by composing the cooperation plan and all of the agents together.

In section 4, we first define some inference rules based on the position-exchange principle for reasoning about mutual belief among agents, and then describe under what condition the mutual belief will have been built among agents. Using those inference rules, we can reason about mutual belief among multiple cooperative agents and show whether cooperation can go on rationally among agents. How to reason about mutual belief among agents while cooperating by using the formal method will be shown in an example.
However, agents believe each other mutually cannot guarantee agents to benefit themselves from cooperation. If a malicious agent is purposing on making others believe it falsely by providing untrue information, cooperation will be led to a disaster. That is to say, it will be meaningless of cooperation if an agent cannot prevent malicious agents from cheating or misbehaving during cooperation.

At the current stage, most researches focus on how to implement security agents to prevent agents from being cheated by malicious ones [18][10]. At the security level, the main goal is to protect agents not to be intruded by malicious ones. However, once an agent is authenticated it will be considered honest, and further any actions taken by the authenticated one will be regarded trusty. Anywhere, that is not completely consistent with our commonsense. For instance, in a virtual-market environment or shopping online [15] [17] [7] [9], an unfaithful player may use some reliable information (maybe stolen from other trusty users) first to pass through the authentication to be allowed to enter the transaction place, and then play maliciously to obtain extra advantages. For example, in a virtual market environment, once entering the market, a buyer may purchase many commodities online even though he cannot afford or does not want to pay them at all. On the other side, a seller may sell things at quite low prices in order to dump his poor quality goods. That is to say, even though a player is authenticated with some security measures, we cannot say the player will behave honestly subsequently.

Since it is not sufficient to prevent agents involved in e-commerce from being cheated by only providing anti-cheating mechanisms at the security level, we must provide additional anti-cheating mechanisms at the cooperation (or transaction) phase during a transaction or an exchange is going on. Our strategy of building an anti-cheating mechanism for agents involved in cooperation is as follows.

First, we extend the definition of expectation for agents by quantifying expectations of agents. For an expectation of an agent, it indicates not only what action the agent expects to happen next but also how to evaluate the benefits that the action will bring. If there are so many actions occurring unexpectedly and the negative benefits overrun some threshold that the agent can endure, the agent will conclude that someone else may be misbehaving over it or the other may be attempting to cheat it to take extra advantages from it. Thus, an agent can use its expectations to reason about cheating to avoid being cheated by others.

Second, correspondingly to the extension of expectations, we extend the standard pi-calculus on two aspects. On one hand, we extend the pi-calculus by redefining an action in the pi-calculus as a pair, in which one element indicates the concrete action as usual and the other represents to what degree the action is expected to occur. On the other hand, we define configurations for all processes to keep track of evolutions of processes [2]. Thus, an agent with expectations can easily be defined in the extended pi-calculus and the total expectations and the traces kept in configurations can be used to reason about cheating based on anti-cheating criteria under different circumstances. By using the extended pi-calculus, we can reason about cheating among agents based on those reduction rules defined for the pi-calculus.

Section 5 defines the syntax and semantics of the extended pi-calculus.

In section 6, we redefine agents with quantified expectations and describe how to reason about cheating by using the formal framework, and we will also give an example to show how an agent use its expectations to reason about other’s cheating behaviours.

In section 7, we will conclude our paper and point out our future direction.

2 The Formal Framework

In this paper, we adopt a process algebra approach, the pi-calculus [14], to formalize agents, plans, and cooperation.

In the pi-calculus, there are only two kinds of entities: processes and channels, where processes are active components of a system and they communicate with each other through ports (or names) connected via channels. In the pi-calculus, all of data, channels and variables are names.

The processes in the pi-calculus have the following forms.

\[
P ::= \sum_{i \in I} P_i \mid P \neq Q \mid !P \mid (\lambda x)P \mid [x = y]P
\]

\[
\pi ::= x(y) \mid \bar{y}x \mid \tau
\]

Where, \(I\) is a finite set. \(\sum_{i \in I} P_i \) represents to execute one of these \(I\) processes and when \(I = \emptyset\) we mark \(\sum_{i \in I} P_i \) as \(\emptyset\), which is inert. \(x(y)\) and \(\bar{y}x\) represent that name \(y\) will be input/output along channel \(x\), respectively, whereas \(\tau\) represents a silent action. \(P \neq Q\) represents the parallel composition of two processes
of $P$ and $Q$. $P$ represents any number of copies of $P$. $(\forall v)P$ introduces a new channel $x$ with scope $P$, where $v$ is the constriction operator. $[x = y]P$ means that process $P$ will proceed only if $x$ and $y$ are the same channel, where $[x = y]$ is the matching operator.

In the pi-calculus, the computation and the evolution of a process are defined by reduction rules. The most important reduction relation is about communication.

$$\overline{\alpha}(z) \cdot P | \overline{\alpha}(z) \cdot Q \rightarrow P \mid Q \setminus \{z\}$$

It means that the process will reduce into the right form after the communication, and meanwhile all free occurrences of $z$ in $Q$ will be substituted with $y$.

For simplicity and clarity, when we define a pi-calculus process, we usually use different names for data, channels and variables, for instance, we let $m$ and $n$ range over data, let $\alpha, \beta$ and $\gamma$ range over channels, and let $x, y,$ and $z$ range over variables. Conventionally, we use $P, Q,$ and $R$ to represent processes.

3 Agents and Their Cooperation

We have pointed out that an agent has its behaviour specification. Though an agent is an active entity that may have pro-activity and act actively, a cooperative agent should take actions complyingly with the global cooperation plan. In this section, we first define cooperation plans formally as the pi-calculus processes. And then we define agents formally and show how to bind agents together to perform the cooperation plan.

3.1 Cooperation Plan

A cooperation plan is always composed of a series of tasks, among which there are some specific relationships to coordinate those tasks’ performing. In general, a cooperation plan can be viewed as a tree, in which nodes are tasks to be allocated to agents; and relationships among tasks can be mapped to relationships among nodes. A cooperation plan can be defined recursively and described in the pi-calculus as follows.

1. The cooperation plan has a hierarchical structure, which is represented as a tree.
2. Any task is corresponding to a node within the plan tree. The global task, $P$, corresponding to the global plan, is the root of the plan tree; and then the plan $Plan = def \ P$.
3. If a task $P$ consists of a set of sub-tasks, $P_1, P_2, ..., P_n$, the node corresponding to the task will have as many child-nodes as the sub-tasks. Then the task $P$ can be defined by composing all of these sub-tasks.

$$P = def P_1 | P_2 | ... | P_n$$

4. Among those sibling nodes, there are two categories of relations. If there is a unary relation over $P_i$ or a binary relation over $P_i$ and $P_j$ ($1 \leq i \neq j \leq n$), $P_i, P_j$, and $P$ may need to be redefined.

4.1. Unary relation: Repetition. It means that the corresponding task needs to be performed many times. And $P_i = redef \ P_i$.

4.2. Binary relations. There are four kinds of binary relations between sibling nodes, serialization, synchronization, sequence, and parallel\(^1\).

4.2.1. Serialization. It means that the performing order of two tasks is not important, but the two tasks cannot be carried on concurrently. And $P_i = redef \ P_i$.

$$P = redef (\alpha P_1 \cdot P_2 \cdot \neg \alpha_0) \cdot P_3 | \cdot | P_n$$

\(^1\) While defining the plan process, we require that serialization relations must be considered first, and then synchronization and sequence; otherwise, deadlocks may be brought into the plan process. For example, consider three sub-processes, $P, Q, R$, among which $P$ and $Q$ must be performed serially and $R$ must be carried on before both $P$ and $Q$. If we do not follow the above convention, we may get process $P \cdot \delta_\rho \cdot P \cdot \neg \nu_\rho \cdot P \cdot \delta_\nu \cdot Q \cdot \nu_\rho \cdot R \cdot \delta_\nu \cdot S_\rho \cdot S_\rho$, where $S_\rho = \rho_\rho \cdot \nu_\rho \cdot S_\rho$. Then, if $Q$ communicates with $S_\rho$ before $P$ has a chance to do so, a deadlock will occur.
Where, \( S_q \) is like a \( pv \) semaphore controller in operating systems.

4.2.2. **Synchronization.** Two tasks with a synchronization relation must be performed at the same time. And

\[
P_i = \text{ref} \ z_y \cdot P_i, \quad P_j = \text{ref} \ z_y \cdot P_j
\]

\[
P = \text{ref} \ (\Sigma_{y=1}^{n} P_i) \cdot P_j \cdot \cdots \cdot P_i \cdot \cdots
\]

4.2.3. **Sequence.** The performing of two tasks should be controlled under a restricted order, i.e., one must precede the other. And

\[
P_i = \text{ref} \ P_i \cdot \delta_y, \quad P_j = \text{ref} \ \delta_y \cdot P_j
\]

\[
P = \text{ref} \ (\forall \delta_y) \cdot P_i \cdot \cdots \cdot P_j \cdot \cdots
\]

4.2.4. **Parallel.** They can even be carried on concurrently. For that case, processes need not to be redefined.

5. There are no any other kinds of nodes or relations within the plan tree except for those defined above.

For example, in an electronic commerce community, a price negotiation procedure can be planned as the repetition of price bargaining between two parties (figure 1).

![Diagram of a price negotiation procedure](image)

**Figure 1.** The plan tree of a price negotiation procedure

In the plan, the bargaining process, which is divided into two sub-processes of price asking and striking, will repeat for any times until both sides make a deal. For the price-asking process, it is divided further into two sub-processes, a process asking a price and then the other waiting for a stroked price. For the price-striking process, it is also divided into two sub-processes, one waiting for a price and then the other striking a price back. Once someone (for instance, the bargaining initiator) thinks the stroked price is acceptable, it can stop bargaining and make a deal.

The plan shown in figure 1 can be expressed in the pi-calculus as follows.

\[
\text{PriceNegotiationPlan} = P_0 = (\forall \delta_y) \cdot P_1 \cdot \delta_y \cdot P_2
\]

\[
P_1 = P_{11} \cdot P_{12}, \quad P_2 = (\forall \delta_y) \cdot P_{21} \cdot \delta_y \cdot P_{22}
\]

\[
P_{11} = (\forall \delta_y) \cdot P_{111} \cdot \delta_y \cdot P_{112}, \quad P_{12} = (\forall \delta_y) \cdot P_{121} \cdot \delta_y \cdot P_{122}
\]

When representing a cooperation plan in the pi-calculus processes, we add some new communicating ports to control the execution of sub-processes so that we could represent relationships within a composition process. Generally, when there are relationships such as serialization, synchronization, and sequence in a system, there may occur deadlocks. Fortunately, by using the formalizing procedure described above, we can get a non-deadlock plan process if there is no deadlock among the plan tree.

**Proposition 1.** If there is no deadlock among the plan tree, the corresponding composition process of the plan will be deadlock free.

The proof is quite simple. As discussed above, we can first eliminate the possibility of a deadlock lying in serialization and synchronization relations. On the other hand, any two synchronized processes cannot

---

2 Synchronization relations are symmetric, so we need only to consider those cases that \( i < j \). Thus deadlocks can be avoided among synchronized nodes.
have sequence relations with another process simultaneously, and *vice versa*. That is to say, any sequence relations and synchronization relations are impossible to bring a cyclic waiting-chain into processes if there is no cyclic waiting-chain occurring in the plan tree. Thus, we can say that the translation described above is deadlock free.

### 3.2 Agent

An agent has actions, knowledge, and beliefs of its own and about others. In a cooperative environment, an agent must undertake some tasks to cooperate with others to perform some cooperation plan. Thus, the agent should behave complyingly with a certain cooperation plan.

We can define an agent as an entity that includes actions, tasks it undertakes, and behaviour specifications consistent with a specific cooperation plan. To represent the behaviour specifications of an agent, we define a function of expectation from actions to *true*/*false* to indicate whether the agent expects to observe a certain kind of response after it takes an action. In general, an expectation can be expressed as that what subsequent actions the agent expects to occur at the current stage. However, to maintain an exact current state requires the agent to keep a great deal of data. So we simplify the definition of the expectations, in which an expectation represents to what action the agent expects to occur subsequently after taking the current action.

In this paper, the symbols $A, B, \ldots$ are always used to denote specific agents. An agent is a 5-tuple.

$$A = (\mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{E}, \mathcal{B})$$

Where, $\mathcal{P}$ is a pi-calculus process to define the behaviour of the agent, $\mathcal{A}$ is an action set, $\mathcal{T}$ is a collection of tasks, $\mathcal{E}$ is $A$’s expectations and defined as a function $\mathcal{E}: \mathcal{A} \times \mathcal{A} \rightarrow \{true, false\}$, and $\mathcal{B}$ is $A$’s beliefs.

Components of Agents can be defined on the pi-calculus formally, in which the action set $\mathcal{A}$ is a set of pi-calculus actions, the task set $\mathcal{T}$ is a collection of pi-calculus processes, and for any process $P \in \mathcal{T}$, if there is an action $\gamma$ such that $P = \gamma \cdot P'$, then $\gamma \in \mathcal{A}$ is held. And $\mathcal{P}$ is a composition of elements in $\mathcal{T}$, i.e., suppose $T_1, \ldots, T_n \in \mathcal{T}$, then $\mathcal{P} = T_1 \mid \ldots | T_n$.

Suppose that $\alpha, \beta \in \mathcal{A}$, then $\mathcal{E}(\alpha, \beta) = true$ means that if the agent $A$ takes action $\alpha$, it will expect that action $\beta$ to happen. Normally, the control of outputting a message is held by the agent itself, and the agent can take output actions without delays. On the contrary, the agent must wait if it wants to input something. In general, we can say that only when an agent is waiting for something does it expect that thing to appear, so we will only define an agent’s expectations on its output actions. That is if $\mathcal{E}(\alpha, \beta) = true$, then $\alpha$ can be either an input or an output, but $\beta$ must be an input action.

Though an agent could have any kinds of expectations if it wishes, we would say that only those expectations useful for performing its assigned tasks are rational. So we will only take those expectations that are consistent with the requirements of performing its tasks into consideration.

Suppose $T_1 \in \mathcal{T}$ and $T_\gamma = \ldots, \alpha, \beta, \ldots$, in which $\alpha$ is either an input action or an output action, and $\beta$ is an input action. Then $\mathcal{E}(\alpha, \beta) = true$.

Suppose again that the agent is assigned two tasks, $P_1$ and $P_2$, which are in sequence relation, i.e., $P_1 = (\ldots, \alpha, \ldots) \cdot \delta_{\alpha}$, and $P_2 = \delta_{\beta} \cdot (\ldots, \beta, \ldots)$. Then $\mathcal{E}(\alpha, \beta)$ may be *true*.

Since each agent has its own behaviour specification, actions, tasks, expectations, and beliefs, $\mathcal{P}, \mathcal{A}, \mathcal{T}, \mathcal{E}$, and $\mathcal{B}$ can be viewed as functions with the domain of agents. In the rest context, we use $\mathcal{P}(A), \mathcal{A}(A), \mathcal{T}(A), \mathcal{E}(A)$, and $\mathcal{B}(A)$ to denote the behaviour specification, the action set, the task set, the expectations, and the beliefs of $A$, respectively; and when $\mathcal{E}(\alpha, \beta) = true$, we will say that $(\alpha, \beta) \in \mathcal{E}(A)$.

In this paper, we will only consider such kind of beliefs as whether an agent trusts others, whether the agent is trusted by others, and so on. For convenience, we mark $x \in \mathcal{B}$ as $A \triangleright x$. Suppose there is a set of agents, $Ag$, and $B \in Ag$, then $A \triangleright B$ means $A$ trusts $B$, whereas $A \triangleright (B \triangleright A)$ means $A$ believes that $B$ trusts $A$ as well.
3.3 Bind Agents into the cooperation plan

The cooperation plan is only a cooperation blueprint or specification of tasks, which does not provide concrete actions or functions to perform those tasks. After cooperation is planned, tasks should be assigned to cooperative agents for carrying out. We require that every node among the plan tree should have an agent to take responsibility for. However, for the root and midst nodes, they are only set up for coordination and may not correspond to any concrete tasks. That is to say, an agent responsible for the root or a midst node will coordinate those agents who undertake those sub-tasks corresponding to the node’s children-nodes. And all leaf nodes are concrete tasks and need to be performed by agents by taking or executing some concrete actions or functions.

For example, if we allocate those tasks shown in figure 1 to a seller agent, S, and a buyer agent, B, for instance, \( P_0, P_1, P_{11}, P_{111}, P_{112}, P_2, \) and \( P_{22} \) to \( S \), and \( P_{12}, P_{121}, P_{122}, \) and \( P_{22} \) to \( B \), in which for each task \( P_i \), the agents will take actions \( T_i \) to carry out it. These two agents \( S \) and \( B \) can be defined as follows.

\[
S = \langle P, A_i, T, E, B \rangle
\]

\[
A_i = \{\alpha p, \alpha x, \alpha p, \alpha y \}
\]

\[
T = \{T_0, T_1, T_{11}, T_{111}, T_{112}, T_1, T_2, T_{12}, T_{121}, T_{122}, T_{22}\}
\]

\[
P = \{T_0, T_1, T_{11}, T_{111}, T_{112}, T_1, T_2, T_{12}, T_{121}, T_{122}, T_{22}\}
\]

\[
E = \{\tau, \alpha x, \alpha y, \alpha x, \alpha y\}
\]

\[
B = \{\}
\]

\[
T_{11} = \text{CalculatePrice}_s(p), \overline{\alpha p}
\]

\[
T_{12} = \alpha x
\]

\[
T_{12} = \alpha y
\]

\[
T_{22} = \alpha y
\]

\[
T_{22} = \overline{\alpha y}
\]

Figure 2. Formal definitions of agents \( S \) and \( B \)

Where, \( \alpha \) and \( \overline{\alpha} \) represent actions “asking a price” and “waiting for an asked price” respectively, \( \alpha y \) and \( \overline{\alpha y} \) represent actions “striking a price” and “waiting for a stroked price”, and \( \overline{\alpha} \) asks “Accept the price or not?” and then \( \alpha y \) waits for the answer. Functions \( \text{CalculatePrice}_s(p) \) and \( \text{CalculatePrice}_b(p) \) are used to calculate a new asking price and a new striking price, respectively. \( \text{MakeADeal} \) is a process for making a deal between the seller and the buyer.

For agent \( S \)’s expectations, they mean that the seller hope that it will receive a response after each round of bargaining and the buyer will acknowledge its any questions. For agent \( B \)’s expectations, the buyer may expect that the bargaining must be initiated by someone else, and after it strikes a price it may hope that the seller asks a new price or makes a deal with it.

To assemble cooperative agents into the cooperation plan, we should connect the abstract plan specification with those concrete implementations of agents’ functions. In the pi-calculus, we can use the following method to achieve that.

First, we can view the tasks occurring in the plan process as pointers and make those pointers point to some concrete actions or functions. For example, suppose that \( P_i \) is a task occurring in the plan process and assigned to agent \( A \), who will undertake that task by taking action \( T_a \), then we can define \( P_i \) and \( A \) as the following processes.

\[
P_i = \overline{z_{iA}}
\]

\[
A = z_{iA} \cdot T_a
\]

Then compose the processes defined above into a composition process, that is

\[
P_i \mid A = \overline{z_{iA}} \mid z_{iA} \cdot T_a
\]

Thus we bind the agent with the plan together.

On the other hand, an agent may undertake several tasks, for instance, \( T_1, T_2, \ldots, T_k \in T(A) \), then \( P(A) \) can be re-defined as a composition of processes.

\[
P(A) = z_A \cdot T_1 \cdot z_2 \cdot T_2 \cdot \ldots \cdot z_k \cdot T_k
\]

Thus, a cooperation system with a cooperation plan, \( Plan \), and a series of cooperative agents, \( A_1, A_2, \ldots, A_n \), can be defined as follows.

\[
Sys = Plan \mid P(A_1) \mid P(A_2) \mid \ldots \mid P(A_n)
\]
4 Reason about Mutual-Belief

In this section, we will define some inference rules for reasoning about mutual-belief among agents. While defining those rules, we mix the position-exchange principle into the definitions. And then we will describe in what condition agents will believe each other mutually.

4.1 Rules on Beliefs

To define rules on beliefs, we should first know what actions are observable to an agent. In the above context, we used formula \( P \xrightarrow{\gamma} P' \) to represent a process evolved into another process. However, actions of an agent are expressed using a set of pi-calculus actions and processes. We cannot use the form \( A \xrightarrow{\gamma} A' \) again to represent an agent evolves into another agent. To represent an agent observes an action, we assign the form \( A \xrightarrow{\gamma} g \) with the following meaning.

After going through some computations, agent \( A \) will reach a new state, in which all of its components will have new values. Suppose that there are a process and a series actions \( \alpha_1, \alpha_2, \ldots, \alpha_k \) such that

\[
P \xrightarrow{\alpha_1, \alpha_2, \ldots, \alpha_k} P', P' \xrightarrow{\gamma} P'', \gamma \neq \tau \land \gamma \in \mathcal{A}(A)
\]

Then \( A \xrightarrow{\gamma} A' \) has the following operation semantics.

\[
A \xrightarrow{\gamma} A'
\]

Where, \( \alpha \) can be any input or output action except the silent action. It means if there is a process which is able to compute further under \( A \)'s new state, the agent will be able to observe the action and evolve into another new state once the process reduces further.

In general, an agent knows nothing about others. To build beliefs on others, it can only base on those messages it has sent and received. Though an agent will act according to its behaviour specification and it can send messages by consulting the specification and then takes for granted that the destination agent will trust it, it cannot control others to respond it as it requests. That is to say, not all messages it receives are something that it is waiting for or expecting, and thus it cannot draw out whether the others trust it indeed based only on those messages it has received. In our definitions of rules on beliefs, we include the expectations of agents into the premises of rules so agents will only believe things that they are expecting.

Based on the position-exchange principle, an agent can derive beliefs on it from messages it receives, and then derive beliefs on others from messages it sends.

1. Belief about honesty of the other

   If the agent receives a message that it is expecting, it will believe that the sender agent is trustable.

   \[
   A \xrightarrow{\alpha} A', \exists \alpha < \alpha, \beta \in \mathcal{E}(A), \beta \in \mathcal{A}(B) \quad A \triangleright B
   \]

   Where, \( \alpha \) can be either an input action or an output action, whereas \( \beta \) must be an input action.

2. Belief on the other’s belief

   Correspondingly, under the position-exchange principle, \( A \) will believe that agent \( B \) also trusts it if \( A \) responds a message to \( B \) as \( B \) requests.

   \[
   A \xrightarrow{\beta} A', \exists \alpha < \alpha, \beta \in \mathcal{E}(B), \beta \in \mathcal{A}(B) \quad A \triangleright (B \triangleright A)
   \]

   While using the position-exchange principle in the above rule, we do not substitute all occurrences of \( A \). Instead, we just replace the action \( \beta \) with its complement one \( \overline{\beta} \) since \( A \) may not be clear how the receiver, \( B \), is evolving.

4.2 Mutual Belief among Agents

The rules on beliefs defined above can be used to reason about an agent’s beliefs on it and on others. As we mentioned above, to cooperate under some cooperation plan, agents should build mutual belief among them. Informally, we say two agents have built mutual belief if both of them trust each other and each of them believes that it counterpart also trusts it. Then, the mutual-belief can be defined formally in several groups of beliefs.

1) Both of the agents believe in its counterpart.
\[ \begin{align*}
A &\triangleright B \\
B &\triangleright A
\end{align*} \]

2) Each of the two agents believes its counterpart trusts it as well.
\[ \begin{align*}
A &\triangleright (B \triangleright A) \\
B &\triangleright (A \triangleright B)
\end{align*} \]

**Definition:** At-Least-Rationality of cooperation. For a cooperation plan, in which its tasks are allocated to some cooperative agents, if those agents cannot build mutual belief during cooperation, we will say that cooperation will not proceed smoothly and it is irrational. In other words, to build mutual belief among agents is the least requirement for cooperation. So if agents can build mutual belief during cooperation, we say that cooperation is at least rational.

### 4.3 Reason about Mutual Belief among Agents – an Example

Consider the example shown in figure 1 again, the complete plan, and part of agent S and agent B are redefined as follows.

\[\text{PriceNegotiationPlan} = \langle((\forall \delta) z_1 \cdot \delta_1 \cdot z_2) \cdot ((\forall \delta_2) z_3 \cdot \delta_2 \cdot z_4) \cdot \delta_3 \cdot ((\forall \delta_4) \delta_4 \cdot \delta_5 \cdot \delta_6) \rangle \]

\[\mathcal{T}(S) = z_1 \cdot P_{111} \cdot z_2 \cdot P_{112} \cdot z_3 \cdot P_{21}, \quad \text{and} \quad \mathcal{T}(B) = z_3 \cdot P_{121} \cdot z_4 \cdot P_{122} \cdot z_5 \cdot P_{22}\]

Then the procedure to reason about mutual belief between S and B can be proceeding at the same time while the computations between S and B are going on.

1. **S** calculates out an asking price and sends it to **B**, and then waits for **B**’s response. On the other side, **B** is waiting for **S** to ask for a new bargaining price. If **B** receives the message from **S**, in other words, **B** observes action \( \alpha \cdot x \) occurring, then it will have the following belief by using the inference rule BR1.

\[ \frac{\alpha}{B \rightarrow B'}, \langle \tau, \alpha \rangle \in \mathcal{E}(B), \alpha \in \mathcal{A}(S) \quad \text{then} \quad B \triangleright S \]

2. Once after **B** receives an asking price, it will calculate a new price for striking and then send it back to **S**. At that case, **B** can generate another belief on **S** by using the inference rule BR2.

\[ \frac{\alpha}{B \rightarrow B'}, \langle \alpha, \alpha \rangle \in \mathcal{E}(S), \alpha \in \mathcal{A}(S) \quad \text{then} \quad B \triangleright (S \triangleright B) \]

On the other side, for **S**, it will have the following belief according to BR1.

\[ \frac{\alpha}{S \rightarrow S'}, \langle \alpha, \alpha \rangle \in \mathcal{E}(S), \alpha \in \mathcal{A}(B) \quad \text{then} \quad S \triangleright B \]

3. By now, **B** has believed that **S** is trustable and it itself is also trustable for **S**. However, **S** is not certain whether it is trusted by **B** or not though it has trusted **B**. If cooperation stops now, the two agents have not built mutual belief between them and thus the cooperation plan would seem uncompleted. Nevertheless, according to the cooperation plan, agent **S** has two choices for its succeeding actions, to go on bargaining with **S** or to accept the striking price and make a deal with **B**.

3.1. Continue bargaining by suggesting another asking price to **B**. Then according to the inference rule BR2, **S** will generate a new belief on **B**.

\[ \frac{\alpha}{S \rightarrow S'}, \langle \alpha, \alpha \rangle \in \mathcal{E}(B), \alpha \in \mathcal{A}(B) \quad \text{then} \quad S \triangleright (B \triangleright S) \]

3.2. Or stop bargaining and make a deal with **B**. Similarly as 3.1, **S** will generate a new belief on **B**, too.

\[ \frac{\alpha}{S \rightarrow S'}, \langle \alpha, \alpha \rangle \in \mathcal{E}(B), \alpha \in \mathcal{A}(B) \quad \text{then} \quad S \triangleright (B \triangleright S) \]

Now, although the computations between **S** and **B** have not finished, the mutual belief has been built between them. If we reason about further, we can only enhance the mutual belief between them. Thus we can say cooperation between **S** and **B** is rational.

### 5 An extension of the pi-calculus

In the following sections, we will study how to build trusty mutual-belief among agents by reasoning about cheating behaviours of malicious agents.

We have mentioned that the expectations of agents should be quantified to express how many benefits agents have obtained. So we should extend the formal framework first to meet the requirements for
quantifying agents’ expectations and reasoning about cheating. We extend the standard pi-calculus on two aspects.

First, we redefine an action occurring in a process as a pair, in which the first element is with the same meaning as that in the standard pi-calculus whereas the second element is a unary function to calculate the expectation degree that the action is expected to occur.

**Definition 1. Action and Expectation function.** Suppose Act is the set of actions, any action in Act has the form \( \langle \alpha, f(\alpha) \rangle \), where \( \alpha \) is an operation of input/output/silence and \( f(\alpha) \) is the expectation and defined as \( f: \text{Act} \rightarrow [-1, +1] \). For any action \( \langle \alpha, f(\alpha) \rangle \in \text{Act} \), suppose that \( f(\alpha) = \omega \), then

1) \( \omega \geq 0 \) means that action \( \alpha \) is expected to occur, and the greater \( \omega \) is, the stronger the expectation is.

2) \( \omega < 0 \) means that action \( \alpha \) is expected not to occur, and the less \( \omega \) is, the stronger the expectation is.

Second, we construct configurations for processes, in which the first element is a trace of actions whereas the second element is a process as usual.

**Definition 2. Trace and Configuration.** Suppose Act* is the set of strings of actions, then a trace \( t \) is a string ranged over Act*, i.e., \( t \in \text{Act}* \). \( \emptyset \) (empty) is a trace, and if \( t_1 \) is a trace \( t_1 \cdot \langle \alpha, f(\alpha) \rangle \) is also a trace. A configuration \( C \) is a pair of a trace \( t \) and a process \( P \), i.e., \( C = \{ t, P \} \).

Since any action in a trace records the expectation degree of itself, we can calculate the total expectation till the current stage of a process’s evolution and then use it to reason about whether the process is satisfied with its evolution.

In the following context of this sub-section, we will first describe the formal syntax of the extended pi-calculus and then give its formal semantics.

### 5.1 Syntax of the extension

Suppose \( A_1, A_2, \ldots, A_n \) are a series of process identifiers, then processes in the standard pi-calculus will change into the following forms.

\[
P := \sum, \pi \cdot A \mid A \mid A \mid !A \mid (\&x)A \mid [x = y]A
\]

Where, \( t \) is the trace of actions that \( A \) has performed till the current stage. Initially, \( t = \emptyset \).

Correspondingly, actions occurring in processes change into the following forms.

\[
\pi := \langle \gamma, f(\gamma) \rangle
\]

\[
\gamma := x(y) \mid [xy] \tau
\]

Where, \( f(\gamma) \) is the expectation function corresponding to \( \gamma \) and for different actions, the expectation functions may be different.

### 5.2 Formal semantics of the extension

Similarly to the standard pi-calculus, the formal semantics of the extended pi-calculus is also defined in reduction relations. Since there is always an expectation function related to an action, for simplicity, we will omit the expectations of actions when we are describing the formal semantics of the extended pi-calculus.

1) **Evolution Rules**

Evolution rules describe how a process evolves after an input/output/silent action and accumulates the effect of the computation of each step.

**INPUT:** \( \{ t, \alpha(x) \cdot P \} \xrightarrow{\alpha(x) \cdot P} \{ t \cdot \alpha(y), P[\gamma]\} \)

If there is an output action related to the input action, the input action will receive the data sent by the output action and replace all of the variable \( x \) with the received data \( y \). \( P[\gamma/x] \) means all free occurrences of \( x \) in \( P \) will be substituted with \( y \). Once after the input, the input action will be added into the trace within the configuration, in which the variable \( x \) will also be replaced with the received data \( y \).

**OUTPUT:** \( \{ t, \alpha(y) \cdot P \} \xrightarrow{\alpha(y)} \{ t \cdot \alpha(y), P \} \)

When outputting, it only needs to record the output action in the configuration.
The silent action does not effect the configuration of a process. That is to say, for internal communications, the communication can be regarded as silent actions and will not put any effects on the configurations of processes.

2) Communication rule

COMM: \( \langle t_0 \cdot \alpha(y) \cdot Q \rangle \xrightarrow{\gamma} \langle t_0 \cdot \alpha(y), Q(\gamma) \rangle \)

While communicating, both of the configurations will evolve independently.

3) Parallel rule

PAR:

\[ \langle t_0, P \rangle \xrightarrow{\gamma} \langle t_0 \cdot \gamma', P' \rangle \]

\[ \langle t_0, Q \rangle \xrightarrow{\gamma} \langle t_0 \cdot \gamma', Q' \rangle \]

Where, \( \gamma = \alpha(y) \) if \( \gamma = \alpha(x) \) and \( \gamma \) is communicating with \( \alpha \), and \( \gamma' = \alpha(y) \) if \( \gamma = \alpha(y) \).

6 Reason about Cheating

To reason about cheating, we should first formalize agents in the formal framework mentioned above, and then use the semantic rules to reason about the expectations of agents. For any agent, it may have its own criteria to judge under what condition someone else may be attempting to cheat it. In general, an agent may have a certain threshold for the satisfaction degree and once the satisfaction degree is beyond what it can endure it will conclude that it is being cheated. In this section, we first describe how to express agents in the formal framework and then show how to use the formalized information to reason about cheating by illustrating an example.

6.1 Extended Agent

In previous section, we have defined an agent as a combination of tasks, expectations, and beliefs, in which expectations have only two values, true or false. However, to reason about cheating, we should know how much the agent expects something to occur, i.e., the expectations need to be quantified. So we extend the definition of an agent by dividing the expectations into two parts, one indicates what actions the agent expect to occur subsequently and the other represents to what degree the action that the agent takes currently satisfies the agent’s expectation. For the first part, we quantify the expectation function by altering its range from \{true, false\} to \([-1 .. +1\]}. For the second part, we use a configuration to record the agent’s behaviours and its satisfaction degree.

Then we can redefine an agent as a 9-tuple.

\[ A = \langle P, A, T, E, \hat{B}, t_0, C, S_{max}, S_{min} \rangle \]

Where, \( t_0 \) is an initial trace that the agent had gone through by now. \( P \) and \( t_0 \) will together form an initial configuration.

\( E \) is a variant of the expectation function \( E \), which is redefined as: \( A \times A \rightarrow [-1 .. +1] \), and \( E(\alpha, \beta) = true \) if \( E(\alpha, \beta) \geq 0 \). \( E(\alpha, \beta) = \omega \) means the agent expects action \( \beta \) to occur after taking action \( \alpha \) and the degree of expectation is \( \omega \).

For any \( E(\alpha, \beta) = \omega \).

1) \( \omega \geq 0 \). It means that the agent expects action \( \beta \) to occur, and the greater \( \omega \) is, the stronger the expectation is.

2) \( \omega < 0 \). It means that the agent expects action \( \beta \) not to occur, and the less \( \omega \) is, the stronger the expectation is.

In the above context, because the pi-calculus is stateless and we can only use information related to the current action, we have defined the expectation function of an action in the extended pi-calculus as a unary function. It seems that the function defined in an action in the extended pi-calculus is incompatible with the expectation function defined for the agent. Nevertheless, we can still use the expectation function of an action to represent the expectations of an agent. Suppose that a process will go through a new action
\langle \beta, f(\beta) \rangle$ just after experiencing a preceding action $\langle \alpha, f(\alpha) \rangle$ and the corresponding expectation of an agent is defined as $E(\alpha, \beta)$, then we can simply let $f(\alpha) = E(\alpha, \beta)$, where $E(\alpha, \beta)$ should have been defined.

$C$ is a set of criteria for judging what behaviours benefit to the agent.

$S_{\text{max}}$ is the maximum satisfaction degree that the agent expects and $S_{\text{min}}$ is the minimum satisfaction degree that the agent can endure. Usually, $S_{\text{min}}$ is a small (even non-positive) value, which means the total expectation of the agent cannot be less than that threshold.

### 6.2 Criteria of judgement

To reason about cheating, an agent should first know how to judge what behaviours are malicious and unacceptable. For different applications, the criteria of judgement may be different. Nevertheless, in the real world, the most valuable thing that people are concerned with is their profit and the most important goal is to gain more benefit or at least not lose their profit. So there are two criteria that must be fitted for most situations.

1) Too bad must be bad, and
2) Too good to be good.

Then we can apply these two criteria to different situations that agents involving in.

Suppose that the maximum satisfaction degree and the minimum satisfaction degree are $S_{\text{max}}$ and $S_{\text{min}}$, respectively, we can define the general rules for reasoning about cheating for an agent as follows.

1) For any configuration of the agent, $C = \{t, P\}$, that $\neg \exists \alpha, \beta \cdot (\overline{C}, \beta \subseteq t \wedge E(\overline{C}, \beta) \geq S_{\text{min}})$. It means that the agent will consider its interlocution is trying to cheat it if its interlocution returns something better by far than it has expected.

2) For any $C = \{t, P\}$, that $\neg \exists \alpha, \beta \cdot (k = i..j) \cdot (\overline{C}, \beta_i, ..., \overline{C}, \beta_j \subseteq t \wedge \sum_{i=j} E(\overline{C}, \beta_k) \leq S_{\text{min}})$. It means when the negative expectations of the agent are being accumulated to some extends that the agent cannot endure the agent will conclude that its interlocution is cheat it by referring to the principle that “too bad must be bad”.

### 6.3 Reason about Cheating Behaviours -- An Example

Consider the price negotiation example mentioned in section 3 again. For the seller and the buyer, they have their own interested benefits. To protect their benefits, they may have different strategies and then have different standards to judge what are not good for them. For instance, the buyer always expects to purchase things at lower prices. Then the buyer may apply the following criteria to evaluating the seller’s behaviours.

a) If the seller is asking a price higher by far than the market price that the buyer knows, the buyer will apply the criterion that “too bad must be bad” and suspect the seller that the seller is trying to earn over-profit from it.

b) On the contrary, if the seller is asking a quite lower price, the buyer will apply the criterion that “too good to be good” and conclude that the seller must be attempting to cheat it.

c) If the seller reduces its asking price so quickly (for instance, at the second round, the seller asks a new price less than half of the first round price after the buyer strikes back a wanting price.), it is reasonable for the buyer to doubt the seller’s motivation because the selling price drops so sharply that it seems to benefit the buyer too much.

d) If the seller asks a high price at first but it agrees to make a deal at a very low price (for instance, less than half of the original price) after a few bargaining rounds, the buyer may also doubt the honesty of the seller because the seller is giving up too much profit.

On the other hand, the seller may have criteria different from the buyer’s to analyze the buyer’s behaviours.

a) Usually, the seller will ask a quite high price at the beginning of a bargaining and wait for the buyer to strike back an acceptable low price. However, if the buyer accepts a high asking price

---

3 At any stage, a process must have gone through some preceding actions (including the silent action $\tau$).
4 To avoid the deadlock of the computation of a process, we always suppose that both $\alpha$ and $\beta$ cannot be the silent action, $\tau$, at the same time.
that even the seller considers a bit higher than the seller has expected, it is too good for the seller
and the seller has reasons to suspect the buyer’s motivation.

b) Contrarily, if the buyer does not raise its striking price to a reasonable one, it is reasonable for
the seller to doubt that the buyer sincerely wants to make a deal.

Then the two agents, $S$ and $B$, corresponding to the seller and the buyer respectively can be redefined
in the extended pi-calculus as follows.

$$S = \langle \mathcal{P}, A, \mathcal{T}, \mathcal{E}, B, t_0, C, 1, -1 \rangle$$

$$T_{111} = \text{CalculatePrice}_s(p), \langle \tilde{a}(p), \mathcal{E}(\tau, \tilde{a}(p)) \rangle$$

$$P_{112} = \langle \tilde{a}(p), \mathcal{E}(\tau(\tilde{a}(p)), \tilde{a}(p)) \rangle$$

$$P_{11} = \langle \tilde{a}(p), 0 \rangle, \langle \tilde{a}(p), \mathcal{E}(\tau(\tilde{a}(p)), y) \rangle \quad \text{if } y = \text{true}$$

MakeADeal

Where, $t_0 = \emptyset$, and the expectations $\mathcal{E} = \{ \mathcal{E}(\tau(\tilde{a}(p))), \mathcal{E}(\tilde{a}(p), \alpha(x)), \mathcal{E}(\tilde{a}(p), \alpha(y)) \}$.

We let the expectation degrees related with some actions be 0, for instance $\mathcal{E}(\tau(\tilde{a}(p))) = 0$, to denote
that $S$ need not pay attention on its own actions of asking prices.

Suppose the purchase price is $S_p$, and $S$ expects to earn profit at most $Max\%$ of the purchase price and
at least not less than $Min\%$ of the purchase price, then the expectation degree for getting a striking price is
listed as follows.

$$\mathcal{E}(\tilde{a}(p), \alpha(x)) = \left\{ \begin{array}{ll}
1, & \text{if } x > p \\
\frac{1}{((max - min)/2)^2} \times (x - (S + \frac{max - min}{2}))^2 + 1, & \text{otherwise}
\end{array} \right.$$
\[ S = \langle \mathbb{P}, \mathbb{A}, \mathbb{T}, \mathbb{E}, \mathbb{B}, t, \mathbb{C}, 1, -1 \rangle \]

......

\[ T_{i1} = \langle a(x), \mathbb{E}(\tau, a(x)) \rangle \]

\[ T_{i2} = \langle \mathbb{E}(\tau, a(x)), \mathbb{E}(\mathbb{O}(p'), a(x)) \rangle \]

\[ - \text{CalculatePrice}_a(p) \]

\[ T_{i3} = \langle \mathbb{O}(x), \mathbb{E}(\mathbb{O}(p'), \mathbb{O}(x)) \rangle \]

\[ \text{CalculatePrice}_a(p) \]

\[ T_{i4} = \langle \mathbb{O}(x), \mathbb{O}(x), \mathbb{O}(x), \mathbb{O}(x) \rangle \]

\[ \text{CalculatePrice}_a(p) \]

Where, \( t_0 = \emptyset \), the expectations \( \mathbb{E} = \{ \mathbb{E}(\tau, a(x)), \mathbb{E}(\mathbb{O}(p'), \mathbb{O}(x)) \} \), and some actions with 0 expectation degree means that \( B \) need not pay attention on its own actions of striking/accepting/rejecting prices.

Suppose that \( B \) knows the average market price is $S$, and it expects to buy the commodity at a price around the average market price, for instance the final price is not higher than the market price at most \( \text{Max}\% \) or lower than the market price at most \( \text{Min}\% \), then the expectation degree for receiving an asking price and suggesting a deal price are listed as follows.

\[
E(\tau, a(x)) = \mathbb{E}(\mathbb{O}(p'), a(x)) = \mathbb{E}(\mathbb{O}(p'), \mathbb{O}(x)) = \frac{1}{((\text{max} - \text{min})/2)^2} \times (x - (S + \text{max} - \text{min})^2 + 1)
\]

Based on the analysis above, the set of benefit criteria \( \mathbb{C} \) can be defined as follows.

a) For any configuration, \( \mathbb{C} = \{t, \mathbb{P}\} \), that \( \neg \exists a \cdot (a \subseteq t \land (E(\tau, a) \geq S_{\text{min}} \lor E(\tau, a) \leq S_{\text{max}})) \). It means \( B \) will suspect the seller if the seller asks a price higher or lower by far than \( B \) has expected.

b) For any \( \mathbb{C} = \{t, \mathbb{P}\} \), that \( \neg \exists a \cdot (a \subseteq t \land \sum_{i=1}^{p} E(\tau, a_i(x)) \leq S_{\text{max}}) \). It means that if the seller cannot provide an acceptable price in a certain rounds of bargaining, \( B \) may think that the seller is not sincere enough to make a deal.

During the transaction, both \( S \) and \( B \) will monitor their input/output actions and analyze those input/output action sequences whether or not satisfy their anti-cheating criteria, respectively. Once they find that some actions violate their expectations for better profit, they will conclude that their interlocutor is cheating.

6.4 A Brief Discussion about the Performance

For the above example, we can find that to reason about cheating behaviours, an agent need only consider two adjacent steps under most situations. That is to say, the performance for an agent to reason about cheating is mostly determined by those reasoning criteria that are related to a sequence of steps (for instance, both of \( c \) for the two agents).

Nevertheless, when we are defining the expectation functions, we select a special kind of function, parabola. When an agent is applying its criterion of \( c \), it need only consider those negative expectations. So it is quite easy for both of the two agents to find a sequence of steps to evaluate the formulas defined in the criteria of \( c \).

7 Conclusions

In [3], it gave three criteria for cooperation among multiple agents. Briefly, to cooperate, all agents must believe each other mutually. However, cooperation schemas in the literature take mutual belief for granted, and they always assume that cooperating agents believe each other mutually, which will leave many chances for malicious agents to do harms on cooperation. Only when we know that every agent participating in cooperation believes each other mutually can we say that cooperation will go through rationally and smoothly.

In this paper, to reason about mutual belief among agents, we adopt a technique using the position-exchange principle. By using those inference rules based on the principle, we can reason about an agent’s
beliefs on it and on others. In [19], a different inference rule was used to reason about knowledge of others. That inference rule can be expressed as follows.

\[ B, B_\alpha (\alpha \rightarrow \beta) \rightarrow (B, B_\alpha \rightarrow B, B_\beta \beta) \]

Intuitively, the rule says that if A believes that B believes some implication is held, then once A believes that B believes the premise of the implication is satisfied then A will also believe that B will believe the result of the implication is implied.

That inference rule has several main differences from ours. First, it requires that A must have already had some beliefs on B. Second, there is no variable substitutions occurring in that rule, which means that the rule can only be applied to the circumstance that all agents have completely common knowledge. However, in a distributed environment, agents are incapable of possessing knowledge and beliefs about others in advance, and it is impossible for agents to maintain all knowledge dispersed within the environment, either, which will lead the above rule unsuitable for real distributed systems.

Before defining the position-exchange principle in inference rules, we first take a process algebra approach, the pi-calculus, to formalize cooperation plans and then define an agent as an entity with actions, assigned tasks, expectations, and beliefs. While defining the inference rules for reasoning about mutual belief among agents, we take an agent’s expectations into consideration and bind the expectations with its beliefs together so that the agent will only believe what it is expecting. Thus once mutual belief is built among agents; we will be able to say that cooperation will go on rationally.

As we have pointed out that agents believe mutually cannot prevent malicious agents from behaving dishonestly while cooperating.

Though many researchers have studied the cheating behaviours of players in e-commerce environment, few of them paid their attentions on anti-cheating at exchange or transaction level.

In this paper, we extended the standard pi-calculus for reasoning about cheating behaviours among agents. First, we quantified an agent’s expectations to describe what actions the agent expect to occur, to what degree the agent expects an action to occur, and to what degree the agent is satisfied with the action’s occurrence. Second, we extend the standard pi-calculus to specify the expectations of agents, in which each action records the expectation degree of the action’s occurrence and each process keeps the trace of actions during the agent’s evolution.

By keeping traces of actions in the pi-calculus processes, we can reason about the history of actions that agents have performed and define criteria of anti-cheating based on those traces. And then agents can use their criteria to reason about cheating by monitoring their actions and analyzing whether their actions violate their criteria.

The extended pi-calculus is much similar to the stochastic process algebra [8][1]. However, in the stochastic process algebra, the rate of probability of an action can only affect the current state of a process and the total performance of a system should be recorded by using some additional mechanism. Contrarily, the new process algebra in this paper can accumulate the effects of all actions the process has gone through and will go through, and then an agent expressed in the pi-calculus can use the total effect to judge whether someone else is trying to misbehave over it.

In addition, the game theory [5] can also be used to reason about competition and cheating. However, the game theory is often used on the situation that there are only two participants and has only one decision step. Differently, to reason about cheating, our process algebra does not care whom an agent’s interlocutors are and how many interlocutors the agent has. Moreover, our process algebra can keep the trace of an agent’s actions and reason on those actions for anti-cheating.

At next stage, we will extend the process algebra further and study how to build an anti-cheating mechanism for the complete exchange procedures in the e-commerce environment.

References: