Reasoning About Belief based on Common Knowledge of Observability of Actions

Hideki Isozaki
NTT Basic Research Laboratories
3-1, Morinosato-Wakamiya
Atsugi-shi, Kanagawa-ken, 243-01, Japan
isozaki@ntt-20.brl.ntt.jp

Abstract

Intelligent agents should regard other agents' thoughts. If they can guess what you think without asking you, they should do so. We human beings notice other people's thoughts without direct communication. One's thoughts depend on one's belief. However, reasoning about belief is very difficult because various factors affect belief and they often lead to inconsistency. This paper presents a simple algorithm to calculate multiagent nested belief from an action sequence. The following three factors are essential in this algorithm: 1) the observability conditions of fluents and actions, 2) the direct/indirect effects of each action, and 3) the incompatibility of fluents. The algorithm performs *regressive* reasoning from a query. It has been tested by dozens of examples through a graphic interface. Experiments show the system gives plausible answers to various queries. This method will be useful in the construction of plan recognition systems and other advanced systems.

Introduction

In order to build intelligent systems such as autonomous robots, intelligent human interfaces, intelligent tutoring systems, agent-oriented programming language interpreters (Shoham 1993), and systems that understand natural language, we have to find a way to make the system guess what others (users or other artificial agents) think. If these systems don't have this ability, they will bother others with useless messages and hazardous behaviors. If they can guess what you think without asking you, they should do so. Even if you don't mind answering their questions, the guess will reduce inapposite questions and save time.

We human beings can guess what other people think without direct communication. One's *location, behavior, and observation* are clues to what one thinks. If we can build a practical algorithm that infers what other agents think, it will be a key technology for various intelligent systems.

This paper presents a simple algorithm which calculates *multiagent temporal nested belief* from these hints. This algorithm uses the *observability of actions* as well as that of *fluents*[1].

We pose the following assumptions for simplicity.

Assumptions. Rules for observability, definitions of actions, and incompatibility among fluents are common knowledge. Each action is performed instantaneously and no actions are performed simultaneously. Every agent learns occurrences of actions only from direct observations. Every agent notices all observable actions and fluents. That is, no one makes an oversight.

In general, reasoning about belief is very difficult and controversial because various factors affect one's belief and they often lead to inconsistency. Belief change is a difficult problem even if there is only one agent (Katsuno & Mendelzon 1992). In multiagent systems, it becomes more difficult because one agent infers another agent's belief from limited information. Most studies on multiagent nested belief (Haas 1986; van der Meyden 1994b; Appelt & Konolige 1988; Kowalski & Kim 1991) introduce belief operators and describe constraints among beliefs. Some studies regard observation (Moore 1985; Konolige 1982; van der Meyden 1994a; Perrault 1990) as well.

However, it is not easy to implement a system based on these studies. They are not very algorithmic and some studies ignore belief changes or assume that the external world is static. As for the *procedural* aspects of multiagent temporal nested beliefs, several methods (Sripada 1993; 1991; Isozaki & Shoham 1992; Isozaki 1993) have been proposed. Although they use time-stamped nested belief data, it is very difficult for an agent to get exact time-stamped nested belief data in a multiagent environment. It is also difficult to maintain these data that correspond to multidimensional regions in a timestamp space.

Here we focus on *up-to-date nested beliefs*, which are usually more important than other complex com-

[1]To represent state changes, temporal reasoning researchers often translate predicates into corresponding terms (reification). A fluent is such a term. It describes a state just like a predicate in the first order logic and a propositional variable in the propositional logic.

binations of temporal beliefs such as John's belief at 10 o'clock about Mary's belief at 12 o'clock about Bob's belief at 8 o'clock (Shoham 1993). The multidimensional time-stamp space (Isozaki & Shoham 1992; Sripada 1991) is too expressive for ordinary applications and causes a lot of problems (Isozaki 1993).

Methodology

This system answers a query by using *domain knowledge* represented by four rule sets (described below). Each query consists of three parts: *whose*, *goal*, and *action sequence*. **Whose** is a list of agents $a_0/\cdots/a_n$ where a_0 = system. It means that the query should be answered by the system's current belief about a_1's current belief about ... about a_n's current belief. The rightmost agent a_n is the *inmost agent*. The integer n is the **depth** of the nested belief. **Goal** is a conjunction of fluents and it may contain variables just as in Prolog. However, you cannot represent quantification inside belief, e.g., John believes that there is *someone* whom Bob loves.

Action sequence gives recent actions that the system should regard. It is a series of actions $\alpha_1, \ldots, \alpha_m$ in chronological order. α_m is the *latest action*. The integer m is the **length** of the sequence. Each action is represented by a ground term that matches an *action rule*'s *head*, which is described later. The overall query means:

"Suppose $\alpha_1, \ldots,$ and α_m occurred in this order. Do you (= the system) believe that a_1 believes that ... that a_n believes that the *goal* holds?"

In short, this algorithm calculates up-to-date nested beliefs based on the latest observation and enhances them by remembering past observations. The system checks from the viewpoint of *whose* if the inmost agent is now observing all fluents in the *goal*. If it is not likely, the algorithm checks from the same viewpoint when the inmost agent observed them last. The inmost agent's observation is calculated using the second inmost agent's belief. Hence, we can decompose deeply nested beliefs into shallower ones.

Domain knowledge

Domain knowledge is represented by four rule sets: $\mathcal{A}$, $\mathcal{X}$, $\mathcal{I}$, and $\mathcal{O}$.

- **Action base** $\mathcal{A}$ is a set of *action rules*. Each action rule indicates the direct effects of an action.
- **Incompatibility base** $\mathcal{X}$ is a set of *incompatibility rules*. Each incompatibility rule represents incompatibility among fluents.
- **Initial database** $\mathcal{I}$ is a set of ground fluents that hold in the initial state of the world. It may be an ordinary Prolog program but it should not return any nonground instances. The database is supposed to have *no* record about other agents' beliefs.
- **Observability base** $\mathcal{O}$ is a set of *observability rules*. Each observability rule indicates a sufficient condition for an agent to observe a fluent or an action.

Although this system has no up-to-date data, it can calculate what is true now by applying the effects of the given action sequence to the initial database. An **action rule** is represented by the formula

head <- *precondition* | *postcondition*.

where *head* is a term that represents the action, *precondition* is a conjunction of fluents that should hold before the action, and *postcondition* is another conjunction of fluents that become true by the action. This rule means that if the precondition holds before the action, the postcondition holds after the action. One can represent the dependency of the postcondition on the precondition and nondeterministic effects of an action by defining two or more rules for the same head.

Although some fluents may become false by the action, they are not explicitly specified in the postcondition, because $\mathcal{X}$ implies which fluents become false. For example, the next rule means that A can grasp X if A is an agent, X is located on $Q, \ldots,$ and X is graspable. This action makes the fluent $\mathsf{on}(X, \mathsf{hand}(A))$ true. At the same time, it makes $\mathsf{on}(X, Q)$ false, but it is not mentioned explicitly in this rule.

$$\begin{array}{l}\mathsf{grasp}(A, X) \texttt{<-} \\ \quad \mathsf{is}(A, \mathsf{agent}), \mathsf{on}(X, Q), \ldots, \mathsf{graspable}(X) \\ \quad \mid \mathsf{on}(X, \mathsf{hand}(A)).\end{array}$$

From now on, we employ DEC-10 Prolog syntax. An identifier with a capital initial letter is a variable name. An identifier with a lowercase initial letter is a constant.

An **incompatibility rule** is represented by the formula

$\mathsf{incomp}(\mathit{fluent}1, \mathit{fluent}2, \mathit{term}1, \mathit{term}2)$.

This means that a ground instance of *fluent1* and a ground instance of *fluent2* are incompatible if they are different and *term1* is identical to *term2*. For example, $\mathsf{incomp}(\mathsf{on}(X, P), \mathsf{on}(Y, Q), X, Y)$ means that if something is located on a certain thing such as a desk, it is not located on any other thing. You might think $\mathsf{incomp}(\mathsf{on}(X, P), \mathsf{on}(X, Q))$ is better than this representation. However, we would like to build a rule which concludes that $\mathsf{on}(A, p_1)$ and $\mathsf{on}(A, p_2)$ are incompatible and that $\mathsf{on}(A, p_1)$ and $\mathsf{on}(B, p_2)$ are not. If we apply the simplified rule to the latter case, A and B are unified and the query succeeds, i.e., the system supposes they are incompatible. Our incompatibility rule avoids this problem. If something is located on a table, the table is not clear. This is represented by $\mathsf{incomp}(\mathsf{clear}(X), \mathsf{on}(Y, Z), X, Z)$. This notation cannot represent arbitrary constraints, but it can represent an important subset of constraints: *functional dependencies* (Ullman 1988).

An **observability rule** is represented by the formula *head* <©> *agent* | *body*.

where *head* is a fluent or an action, *agent* is an observing agent, and *body* is a conjunction of fluents. This means that the *agent* can observe[2] the ground instance of *head* if the instance and the *body* hold. The observability condition of an action is represented by a conjunction of fluents that should hold *before* the action. If there is no observability rule for a certain fluent (or action), nobody can observe it.

For example, the next rule means that agent O can see agent A grasping object X if both A and O are located on the floor of the same room R. One can register two or more rules for the same head.

$$\mathsf{grasp}(A,X) <\!\copyright\!> O \mid \mathsf{on}(A,\mathsf{floor}(R)),\mathsf{on}(O,\mathsf{floor}(R)).$$

Algorithm

Here we describe a simple algorithm that checks if a ground instance of the *goal* holds in *whose*'s belief after the *action sequence*. *Goal* can be decomposed into fluents. For each fluent f, the system generates the term $\hat{f}$, which can be incompatible with f according to $\mathcal{X}$. For example, if f is $\mathsf{on}(a,p)$, then the system gets an alternative fluent pattern $\mathsf{on}(a,Q)$ as $\hat{f}$ by applying $\mathsf{incomp}(\mathsf{on}(a,p),F,X,X)$. If $\mathcal{X}$ has no incompatibility rule for f, the system uses f as $\hat{f}$. Then the system searches a ground instance of $\hat{f}$ by the following algorithm and compares the result g with f. If the system fails to find such an instance, the system answers *unknown*. If g matches f, it answers *yes*. And if f contains variables, the system shows g. If g doesn't match f, the system answers *no* and shows g.

If f is $\mathsf{on}(A,p)$, $\mathcal{X}$ gives $\mathsf{on}(A,Q)$ as $\hat{f}$. However, a ground instance of $\mathsf{on}(A,Q)$ such as $\mathsf{on}(b,q)$ has nothing to do with $\mathsf{on}(a,p)$. In fact, they don't follow the incompatibility rule for on. Hence the system checks $\mathcal{X}$ again when it finds a ground instance.

To realize the aforementioned idea, we introduce the following three factors which affect one's belief.

- **Observation** a_n believes g because g holds and a_n can observe g. This is necessary to simulate other agents' belief change caused by their observations.
- **Causation** a_n believes g because it observed the latest action α_m and a_n knows that α_m causes g.
- **Memory** α_m didn't cause or uncover any ground instance of $\hat{f}$ but a_n believes g because a_n believed g before α_m.

We assume *observation* has higher priority than *causation*, and *causation* has higher priority than *memory*. Sometimes *observation* and *causation* give the

[2]The inference system assumes that O can observe the *head* completely. However, one might want to introduce some ambiguity into observations. For example, O will notice that A is grasping something but it might be impossible for O to identify what is being grasped. In this paper, we ignore such observations.

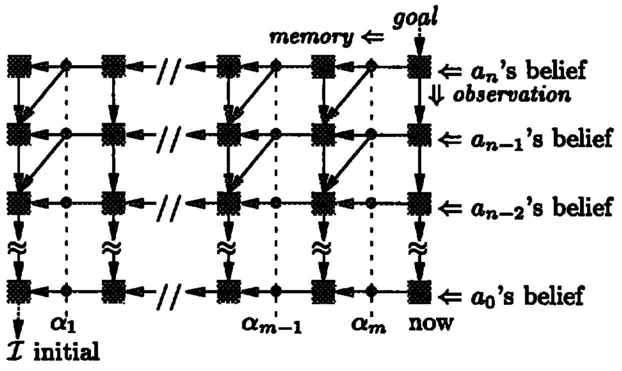

Figure 1: Recursive invocation of the algorithm

same information, but you cannot omit one of them. *Causation* is useful when α_m is observable but g is not. For instance, when you make a phone call, you cannot directly hear the phone ringing, but you believe it is. *Observation* is useful when α_m does not cause g but uncovers g which a_n couldn't observe before α_m. For instance, if you enter a room, you will find various things there. The action doesn't cause the things to be there, but uncovers them.

Now we get the following algorithm. Figure 1 shows how the algorithm works. Each black square corresponds to one's temporal belief. The tiny black circles represent actions.

- If both m and n are zero, none of the above factors affects the result, because it is a query about the system's initial belief. The system generates $\hat{f}$ and finds g from $\mathcal{I}$.
- If $m > 0$ and $n = 0$ hold, the system omits *observation* because it is a query about the system's current belief. First, it checks *causation*. If *causation* fails, it checks *memory*.
- If $m = 0$ and $n > 0$ hold, the system omits *causation* and *memory*. It checks only *observation*.
- If $m > 0$ and $n > 0$ hold, the system has to regard all of the three factors. First, it checks *observation*. If *observation* fails, it checks *causation*. If it also fails, it checks *memory*.

We will describe the last case ($m > 0$ and $n > 0$) in detail. The system checks *observation* as follows. If a_{n-1} believes that g holds now and that a_n can observe g, then a_{n-1} believes that a_n believes g. Therefore, the system searches a ground instance g of $\hat{f}$ that $a_0/\cdots/a_{n-1}$ believes. If such g is found, the system checks whether $a_0/\cdots/a_{n-1}$ believes that a_n can observe g. $\mathcal{O}$ gives a sufficient condition for the observation. Thus the system examines various beliefs of $a_0/\cdots/a_{n-1}$. The system can check these beliefs by using the algorithm with *whose* $= a_0/\cdots/a_{n-1}$. The solid arrows pointing straight down in Fig. 1 represent such recursive calls.

If this *observation* check fails, the system checks *causation*: the direct effects of α_m. The system checks whether g is found in the postcondition of an action rule of the latest action α_m. If such g exists, observability of α_m is checked. Each slant arrow pointing left in Fig. 1 indicates this check. If α_m turns out to be observable for $a_0/\cdots/a_{n-1}$, the algorithm returns g. Otherwise, this *causation* check fails.

However, there is a slight difference between observability check of a fluent and that of an action. As for fluents, the system checks if a given fluent holds from the viewpoint of $a_0/\cdots/a_{n-1}$ when the system checks a_n's observability of the fluent. This is achieved by a recursive call of this algorithm. As for actions, the system should check if α_m occurs from the viewpoint of $a_0/\cdots/a_{n-1}$ when the system checks a_n's observability of α_m. If a_{n-1} doesn't notice the occurrence of α_m, a_{n-1} will not regard α_m to guess a_n's belief. Since each agent learns the occurrence of α_m only from its direct observation, the system should confirm observability of α_m for $a_0/\cdots/a_{n-1}$.

If neither *observation* nor *causation* gives g, the system checks *memory*. Since we know α_m has no effect on f, α_m can be ignored. Therefore, the system calls the algorithm with *action sequence* = $\alpha_1,\ldots,\alpha_{m-1}$. Each arrow pointing straight left in Fig. 1 indicates such a recursive call. This search is repeated until the system finds g or *the action sequence* becomes empty. Even if the sequence is empty, the initial state might have such g and a_n might have observed g. Hence g and its observability are checked in the initial state, too.

Figure 2 is a pseudo code for this algorithm. The function belief returns ground instances in a list if it is given *fluent* (f), *whose*, and *action sequence* (*act-seq*). The function initdb calls $\mathcal{X}$ and $\mathcal{I}$ to answer queries about the system's initial belief. All alternative ground fluents are returned in a list. The function caused searches the postcondition of the latest action for g. The function observable checks if a fluent or an action is observable for *whose*. The function enqueue adds the first argument to the end of the second argument. The function lastelem returns the last element of the given list, and butlast returns the given list minus the last element. The function obscond returns the list (disjunction) of the bodies of all observability rules for the given fluent or action. The function postcond returns the list (disjunction) of the postconditions of all action rules applicable to the given action. The function satisfied checks if the given condition holds by applying belief to each fluent in it. The function pop removes the first element from the specified list and returns the first element. The function action checks if the argument is an action or not. The function instance checks if the first argument is a ground instance of the second argument. The function choiceset returns the list of all patterns of $\hat{f}$.

Improvement of efficiency

The recursive algorithm (Fig. 1) contains redundant computation. There are only $O(mn)$ edges in the figure, but even if we ignore slant edges, there are ${}_{m+n}C_m$ different paths from the top right corner (the original query) to the left bottom corner (i.e. the system's initial belief ($\mathcal{I}+\mathcal{X}$)). Hence the system sometimes calculate the same query more than once. We have implemented a few *progressive* reasoning systems (Isozaki 1994b; 1994a) that are not redundant in this respect. However, they compute beliefs about irrelevant fluents.

The current system records the computed results during the processing of a query and simplifies the query list before its computation. Suppose the system has found that $\mathsf{on}(b,\mathsf{hand}(p))$ is true and is going to find Y and Z such that both $\mathsf{on}(Y,Z)$ and $\mathsf{on}(b,\mathsf{hand}(Y))$ are true. According to $\mathcal{X}$, Y should be p. Hence the simplified query is $\mathsf{on}(p,Z)$.

Since lengthy proofs often have little meaning to users, we add two assumptions to skip them.

> **Extra assumptions for efficiency** Each agent can observe the location of itself. An agent who executed an action can observe the action.

Observation and *causation* are redundant when the latest action uncovers nothing new. To remove this redundancy, the implemented system checks *observation* only for the latest state, the initial state, and intermediate states immediately after the actions that will lead to discovery. We say that such actions **satisfy discovery condition**. The implemented system assumes that only the following actions lead to discovery.

- The inmost agent moved. In this case, the agent finds various things in the new room.
- The inmost agent a_n saw someone else (denoted as a_e) entering the room where a_n is. In this case, a_n finds something that a_e holds in his hand.

The above algorithm omits a precondition check. It is easy to add one to the algorithm, but it leads to unexpected failures because someone in $a_0,\ldots,a_n$ is ignorant about the precondition of an action. When you receive a phone call from a friend, you don't always know where it is. It is almost impossible to enumerate its possible locations. People don't pay much attention to preconditions of actions performed by others.

Results

The author implemented this algorithm in Quintus Prolog and Tcl/Tk. The implemented system Nebula has been tested using dozens of examples through a graphic interface. For example, Fig. 3 shows a map of an example initial state of the world. The long rectangle at the bottom is a hallway. The six large rectangles and inverted L-shaped regions in them indicate booths and desks. Tiny rectangles indicate books. You can

```
function belief(fluent, whose, actseq) {
  if (depth(whose)==0) {/* System's belief */
    if (length(actseq)==0) {return initdb(fluent);
    } else {/* Causation */
      ans := caused(fluent, actseq);
      if (ans ≠ "unknown") {return ans;
      } else {/* Memory */
        return belief(fluent, whose, butlast(actseq));
      }
    }
  } else {/* Nested belief */
    answers0 := belief(fluent, butlast(whose), actseq);
    answers := {};
    foreach g ∈ answers0 {/* Observation */
      if (observable(g, whose, actseq)) {
        enqueue(g, answers); }}
    if (answers ≠ {}) {return answers;
    } else if (length(actseq) > 0) {/* Causation */
      ans := caused(fluent, actseq);
      if (ans ≠ "unknown") {
        /* Observation of the action */
        if (observable(lastelem(actseq), whose,
        butlast(actseq))) {return ans;}
      } else {/* Memory */
        return belief(fluent, whose, butlast(actseq));
      }
    }
    return "unknown";
  }
}

function observable(fluact, whose, actseq) {
  condlist := obscond(fluact, lastelem(whose));
  while (condlist ≠ {}) {cond := pop(condlist);
    if (satisfied(cond, butlast(whose), actseq)) {
      if (action(fluact)) {
        return observable(fluact, butlast(whose), actseq);
      } else {return 1;}}
  }
  return 0;
}

function caused(fluent, actseq) {
  alts := choiceset(fluent);
  while (alts ≠ {}) {incomp := pop(alts);
    postlist := postcond(lastelem(actseq));
    while (postlist ≠ {}) {post := pop(postlist);
      while (post ≠ {}) {ans := pop(post);
        if (instance(ans, incomp) ≠ 0) {return ans;}}}
  }
  return "unknown";
}
```

Figure 2: Checking if *whose* believes *fluent* after *actseq*

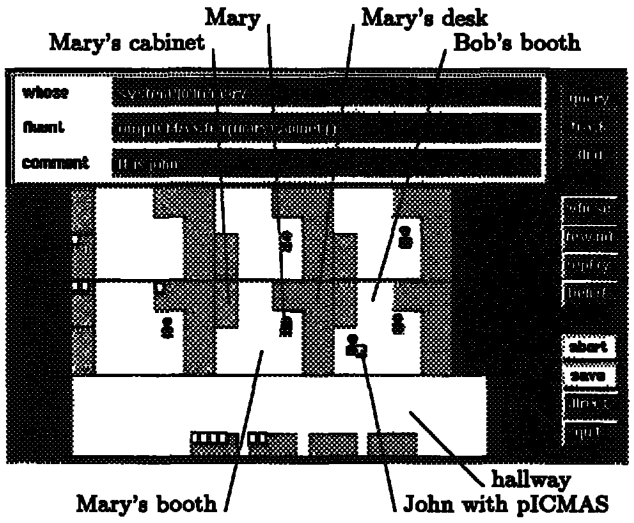

Figure 3: A graphic user interface for the inference system

enter actions by dragging and clicking items on this map. Agents in a booth can see anything in the booth but nothing in other booths. Given actions up to now are shown in another window.

Suppose John is in Bob's booth and Mary is in her booth (Fig. 3). John is holding pICMAS, a copy of the proceedings of ICMAS, in his hand.

First, John moves to the hallway.

α_1 = move(john, floor(hallway))

Second, John enters Mary's booth.

α_2 = move(john, floor(maryBooth))

Third, John puts pICMAS on her desk.

α_3 = put(john, pICMAS, maryDesk)

Fourth, Mary moves to the hallway.

α_4 = move(mary, floor(hallway))

Fifth, John grasps pICMAS again.

α_5 = grasp(john, pICMAS)

Sixth, John puts it on her cabinet.

α_6 = put(john, pICMAS, maryCabinet)

You know pICMAS is now on Mary's cabinet. However, Mary had left her booth before John's moved pICMAS. Let's ask the system whether John believes that Mary believes pICMAS is on her cabinet. This query is denoted as $F(6,2)$ because $m = 6$ and $n = 2$.

($F(6,2)$) whose = system/john/mary;
actionsequence = $\alpha_1, \ldots, \alpha_6$;
goal = on(pICMAS, maryCabinet)

Since $m > 0$ and $n > 0$, the system regards all of the three factors: *observation*, *causation*, and *memory*. Figure 4 shows how the system solves this query.

Mary's observation If John believes that pICMAS is on Mary's cabinet and that Mary can observe pICMAS on her cabinet, John believes that Mary believes that pICMAS is on her cabinet. According

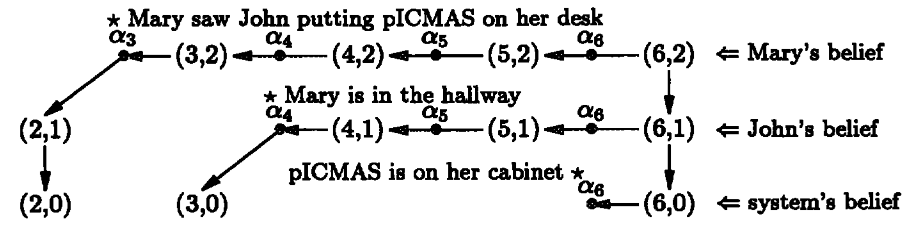

Figure 4: How the system solves $F(6,2)$

to $\mathcal{X}$, on(pICMAS, R_0) is possibly incompatible with on(pICMAS, maryCabinet). The system tries to find R_0 such that John believes that on(pICMAS, R_0) holds now. This can be represented by the following query. From now on, we omit the obvious "slot names" in the queries. Later, the system checks if John believes that Mary can observe the fact (see Mary's observability).

($F(6,1)$) system/john; $\alpha_1, \ldots, \alpha_6$; on(pICMAS, R_0)

$F(6,1)$ also requires all of the three factors. Just as above, the system tries to find R_1 such that on(pICMAS, R_1) holds now. Later, the system checks if John can observe the fact (see John's observability).

($F(6,0)$) system; $\alpha_1, \ldots, \alpha_6$; on(pICMAS, R_1)

Now n is zero; the system regards only *causation* and *memory*. Since α_6 leads to on(pICMAS, maryCabinet), the system gets R_1 = maryCabinet.

John's observability According to $\mathcal{O}$, John can observe pICMAS on her cabinet if John and her cabinet are on the floor of the same room. This condition is denoted as $O(6,0)$ because this query checks *observability*.

($O(6,0)$) system; $\alpha_1, \ldots, \alpha_6$;
on(john, floor(R_2)), on(maryCabinet, floor(R_2))

This query is answered positively because α_2 leads to on(john, floor(maryBooth)) and $\mathcal{I}$ gives on(maryCabinet, floor(maryBooth)). This means that John observes and believes on(pICMAS, maryCabinet). Hence the system gets R_0 = maryCabinet for $F(6,1)$.

Mary's observability (Part 1) According to $\mathcal{O}$, Mary can observe the fluent on(pICMAS, maryCabinet) if Mary and her cabinet are on the floor of the same room.

($O(6,1)$) system/john; $\alpha_1, \ldots, \alpha_6$;
on(mary, floor(R_3)), on(maryCabinet, floor(R_3))

Hence the system checks John's belief about Mary's location.

($F(6,1)'$) system/john; $\alpha_1, \ldots, \alpha_6$; on(mary, floor($R_3$))

The system checks John's *observation*.

($F(6,0)'$) system; $\alpha_1, \ldots, \alpha_6$; on(mary, floor($R_4$))

Since the latest action affecting Mary's location is α_4, the system gets R_4 = hallway, but John cannot observe her because he is in Mary's booth and Mary is in the hallway. Thus, the *observability* check of $F(6,1)'$ fails. Then the system checks *causality*, but α_6 doesn't affect Mary's location. Hence the system checks *memory* by removing α_6.

($F(5,1)'$) system/john; $\alpha_1, \ldots, \alpha_5$; on(mary, floor($R_3$))

Since α_5 doesn't satisfy the *discovery condition*, John's *observation* must be useless. The *causation* check of α_5 is also fruitless.

($F(4,1)'$) system/john; $\alpha_1, \ldots, \alpha_4$; on(mary, floor($R_3$))

Since α_4 doesn't satisfy the *discovery condition*, John's *observation* is omitted. As for *causation*, α_4 causes on(mary, floor(hallway)). According to $\mathcal{O}$, John must have observed α_4 if John and Mary were on the floor of the same room immediately before α_4.

($O(3,0)$) system; $\alpha_1, \alpha_2, \alpha_3$;
on(mary, floor(R_4)), on(john, floor(R_4))

The query is answered positively because $\mathcal{I}$ gives on(mary, floor(maryBooth)) and α_2 leads to on(john, floor(maryBooth)).

Mary's observability (Part 2) The success of $O(3,0)$ means John must have observed α_4 and believe that α_4 caused on(mary, floor(hallway)). Hence the system returns to $O(6,1)$ with R_3 = hallway. Then the goal of $O(6,1)$ is simplified as on(maryCabinet, floor(hallway)). However, the system finds an incompatible fluent on(maryCabinet, floor(maryBooth)) because he knows her cabinet is in her booth. That is, John believes on(pICMAS, maryCabinet), but he doesn't believe that Mary can observe this fact.

Mary's memory Then the system begins to check *causality* and *memory* for $F(6,2)$. As for *causality*, α_6 affects the location of pICMAS but Mary cannot observe α_6. Therefore, the system checks her *memory* by dropping α_6. Since α_5 doesn't satisfy the *discovery condition*, Mary's *observation* immediately after α_5 is omitted. Then the system checks *causation* of α_5. α_5 affects the location of pICMAS, but she cannot observe it. The system drops α_5. Since the innermost agent (= Mary) moved, α_4 satisfies the *discovery condition*, but she observed not her booth but the hallway; it doesn't affect her belief about pICMAS. α_4 is dropped. Since α_3 doesn't satisfy *discovery condition*, Mary's *observation* is omitted. As for *causation*, α_3 affects the location of pICMAS. Mary can observe α_3 if Mary and her desk are on the floor of the same room immediately before α_3.

$(O(2,1))$ system/john; α_1, α_2;
on(mary, floor(R_5)), on(maryDesk, floor(R_5))

The query is answered positively because $\mathcal{I}$ has on(mary, floor(maryBooth)) and on(maryDesk, floor(maryBooth)). Therefore, the system succeeds in proving that John believes that Mary observed α_3 which leads to on(pICMAS, maryDesk). Therefore, the system answers *no* to $F(6,2)$ and shows on(pICMAS, maryDesk). In the same way, we can show that the algorithm answers appropriately to various queries.

However, we found some examples for which Nebula answers inappropriately. Suppose Mary is in the hallway and sees John moving from her booth to the hallway. It is natural to assume that she believes that he was in her booth and that she saw her desk in her booth and so on. However, move(john, maryBooth) doesn't tell that he came from her booth and Mary cannot see John while he is in her room. Therefore, the system concludes that Mary doesn't know where he was.

Discussion

In the previous section, we described an example to which the system answers improperly. It is possible to make the system answer properly to this specific problem without changing its basic framework. For example, we represent the action as move(john, hallway, maryBooth), and modify the algorithm to check where he was before the action. This approach will work for similar problems. However, if we think about the consistency of beliefs about the past, this solution is not sufficient.

Sometimes the given action sequence is not consistent if we think of preconditions. To make the sequence consistent, we have to assume some unmentioned actions and fluents (abduction). In general, there are a lot of consistent combinations of them. If the system can assume unmentioned things freely, the system will lose almost all meaningful nested belief because it will find exceptional histories for them. One of our previous systems checks preconditions and makes assumptions when it encounters a fluent whose truth value is unknown (Isozaki 1994a). It has an awkward mechanism to maintain precondition assumptions. We will have to introduce a more sophisticated assumption maintenance system (Forbus & de Kleer 1993).

This algorithm doesn't assume any policy for the interpolation and assumes persistence as long as they are consistent with later observations. Even if we select a certain interpolation policy, we don't think we have to change the algorithm very much because the interpolation affects only the action sequence and the initial state.

In general, reasoning about belief in a multiagent environment is a difficult problem because it depends on multiagent belief revision (van der Meyden 1994b) and update (van der Meyden 1994a). Even in a single agent case, rational belief change is a difficult problem (Katsuno & Mendelzon 1992). In multiagent environments, an agent has to think of other agents' belief change from its partial knowledge about the world.

This paper presented a practical approach to solving this problem. This method exploits the incompatibility of fluents and the observability of fluents and actions. Kowalski and Sergot (Kowalski & Sergot 1986; Kowalski 1992) also used incompatibility in the Event Calculus. The Event Calculus checks whether the given fluent is true or not at a given point in a given history by finding events that start or end the fluent and confirming no other events interrupt the interval between them. Sripara (Sripada 1988; 1991; 1993) extended the Event Calculus to deal with temporal nested belief. Each rule in his meta-logic system is a nested rule and it is associated with a time interval that indicates its validity. Using time-stamped data, his system and our previous one (Isozaki & Shoham 1992; Isozaki 1993) can compute nested belief about the past.

However, in many applications where nested belief is important, it is difficult to get exact time-stamps and to keep the consistency of beliefs about the past. Usually people don't pay much attention to this consistency because reasoning about the current world and the near future is much more important.

In this paper, we assumed that each agent learns occurrences of actions only from direct observation, and didn't explain how to deal with utterances. Since Speech Act theory (Appelt & Konolige 1988; Perrault 1990; Cohen & Perrault 1979) regards utterances as actions, it is natural to extend this algorithm to cover them. However, utterances don't change the physical world at all and interpolations by mental actions seem to play an important role. In a previous report (Isozaki 1994b), we incorporated notification of ignorance (i.e., "I don't know whether p is true or not.") into an aforementioned progressive reasoning system to solve a variant of the three wisemen puzzle (Kowalski & Kim 1991). It was extended to cover a few other assertions (Isozaki 1994a).

Finally, we have to give formal semantics for this algorithm and to tune it for more useful and practical applications.

Summary

This paper presented an algorithm that computes multiagent nested belief from an action sequence by using common knowledge of observability conditions. This algorithm is simple but replies plausible answers for various queries that are common in everyday life. It doesn't require any nested belief data since it guesses nested belief from the system's belief. We also discussed its problems and extensions. Since this method provides a way to guess what users, friends, and adversaries think, it will be useful in the construction of various advanced systems.

Acknowledgments

The author would like to thank Hirofumi Katsuno, Akihiro Umemura, and the anonymous referees for their useful comments on this paper.

References

Appelt, D. E., and Konolige, K. 1988. A nonmonotonic logic for reasoning about speech acts and belief revision. In Reinfrank, M.; de Kleer, J.; Ginsberg, M. L.; and Sandewall, E., eds., *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, 164–175. Springer-Verlag.

Cohen, P. R., and Perrault, C. R. 1979. Elements of a plan-based theory of speech acts. *Cognitive Science* 3:177–212.

Forbus, K. D., and de Kleer, J. 1993. *Building Problem Solvers*. MIT Press.

Haas, A. R. 1986. A syntactic theory of belief and action. *Artificial Intelligence* 28:245–292.

Isozaki, H., and Shoham, Y. 1992. A mechanism for reasoning about time and belief. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 694–701.

Isozaki, H. 1993. Mutual reasoning based on temporal belief maps (in Japanese). In *Multi Agent and Cooperative Computation II, Lecture Notes in Software Science 5*, 9–27. Kindaikagakusha.

Isozaki, H. 1994a. Analysis of action sequences and birth, growth, and death of nested beliefs (in Japanese). The Forth Workshop on Multi-Agent and Cooperative Computation.

Isozaki, H. 1994b. Reasoning about knowledge in transaction logic (in Japanese). In *JSAI SIG notes on FAI*, 25–32.

Katsuno, H., and Mendelzon, A. O. 1992. On the difference between updating a knowledge base and revising it. In *Belief Revision*. Cambridge University Press.

Konolige, K. 1982. A first-order formalization of knowledge and action for a multi-agent planning system. In *Machine Intelligence 10*. Ellis Horwood Limited. 41–72.

Kowalski, R., and Kim, J.-S. 1991. A metalogic programming approach to multi-agent knowledge and belief. In *Artificial intelligence and mathematical theory of computation*. Academic Press.

Kowalski, R., and Sergot, M. 1986. A logic-based calculus of events. *New Generation Computing* 4:67–95.

Kowalski, R. 1992. Database updates in the event calculus. *Journal of Logic Programming* 12:121–146.

Moore, R. 1985. Semantic considerations on nonmonotonic logic. *Artificial Intelligence* 25:75–94.

Perrault, C. R. 1990. An application of default logic to speech act theory. In Cohen, P. R.; Morgan, J.; and Pollack, M. E., eds., *Intentions in Communication*. MIT Press. chapter 9, 161–185.

Shoham, Y. 1993. Agent-oriented programming. *Artificial Intelligence* 60:51–92.

Sripada, S. M. 1988. A logical framework for temporal deductive databases. In *Proceedings of the 14th conference on VLDB*, 171–182.

Sripada, S. M. 1991. *Temporal Reasoning in Deductive Database*. Ph.D. Dissertation, University of London.

Sripada, S. M. 1993. A metalogic programming approach to reasoning about time in knowledge bases. In *Proceedings of International Joint Conference on Artificial Intelligence*, 860–865. Morgan Kaufmann.

Ullman, J. D. 1988. *Principles of database and knowledge-base systems*, volume 1. Computer Science Press.

van der Meyden, R. 1994a. Common knowledge and update in finite environments. I (extended abstract). In *Proceedings of the Fifth Conference on Theoretical Aspects of Reasoning About Knowledge*, 225–242. Morgan Kaufmann.

van der Meyden, R. 1994b. Mutual belief revision. In *Proceedings of the Fifth Conference on Principles of Knowledge Representation and Reasoning*, 595–606. Morgan Kaufmann.