Remote Belief: Preserving Volition for Loosely-Coupled Processes

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Master of Science in the

Graduate School of The Ohio State University

By

Nuh Aydin, B.S., M.A., M.S., PhD.

* * * * *

The Ohio State University

2002

Master's Examination Committee:

Professor Paolo Sivilotti, Adviser

Professor Neelam Soundarajan

Approved by

____

Adviser

Department of Computer and Information Science

ABSTRACT

Knowledge has proven to be a useful and fundamental formalism for distributed systems. The application of this formalism, however, entails a loss of volition on the part of processes about which something is known. This loss of volition is not appropriate in systems where processes are loosely coupled.

In this thesis we show that it is possible to generalize the notion of knowledge in distributed systems to that of belief. We define the notion of belief formally, derive a number of its properties and show that the notion of belief is more flexible than knowledge, allowing processes to maintain volition. Therefore, the formal semantics of some distributed applications are more accurately described using this new and more general notion of belief, rather than knowledge.

Dedicated to our children Fatima Betul and our second child to come shortly after this thesis is completed

ACKNOWLEDGMENTS

I would like to take this opportunity to thank everyone who has ever helped me throughout my M.S. journey in CIS, including those whose names are not mentioned here.

I am most grateful to my adviser Professor Paul Sivilotti. My interest in the subject started with a course he taught in Spring 2001, which I enjoyed greatly. He has been extraordinarily stimulating, supportive, patient and helpful not only in matters related to research but even with formalities and other problems of life I had to deal with. This thesis would not have been possible without his great help and support.

I am also thankful to Professor Soundarajan for serving in the committee and teaching CIS 755 with a great enthusiasm.

My special thanks also go to Scott M. Pike in CIS Department and Atabey Kaygun in Mathematics Department of The Ohio State University for their intelligent discussions and suggestions.

VITA

September 5, 1972	Born - Istanbul, Turkey
1994	B.S. in Mathematics Education, Middle East Technical University, Ankara, Turkey.
1996	M.A. in Mathematics Education, The Ohio State University, Columbus, Ohio.
1997	M.S. in Mathematics The Ohio State University, Columbus, Ohio.
2002	Ph.D. in Mathematics The Ohio State University, Columbus, Ohio.
February 94-December 94	Graduate Teaching Associate, Middle East Technical University.
September 96 - March 2002	Graduate Teaching Associate, The Ohio State University.
March 2002-present	Lecturer in Mathematics Department, The Ohio State University.

PUBLICATIONS

Research Publications

Siap, I., Aydin, N. and Ray-Chaudhuri D. K. "New 1-Generator Quasi-Twisted Codes over GF(5)." *Codes and association schemes* (Piscataway, NJ, 1999): 265–275, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 56, Amer. Math. Soc., Providence, RI, 2001.

Siap, I., Aydin, N. and Ray-Chaudhuri D. K. "New ternary quasi-cyclic codes with better minimum distances." *IEEE Trans. Inform. Theory* 46 (4): 1554-1558. 2000.

Aydin, N., Siap. I. and Ray-Chaudhuri D. K. "The Structure of 1-Generator Quasi-Twisted Codes and New Linear Codes." *Des. Codes and Cryptogr.*, 24(3): 313-326, 2001.

Aydin, N. and Ray-Chaudhuri D. K. "Quasi-cyclic codes over $\mathbb{Z}_4$ and some new binary codes." to appear in *IEEE Trans. Inform. Theory* July 2002.

FIELDS OF STUDY

Major Field: Mathematics

Studies in:

Algebraic Coding Theory
Computer Science
Mathematics Education

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Although the study of knowledge, or epistemology, has a long tradition in philosophy, the idea of a formal logical analysis of reasoning about knowledge is somewhat more recent. The first book-length treatment of epistemic logic is Hintikka's seminal work, *Knowledge and Belief* [15]. More recently, reasoning about knowledge found applications in such diverse fields as artificial intelligence [8, 19], game theory [1], economics [21], linguistics [4] and computer science [3, 14, 12, 13, 17, 7]. Although the models and contexts are not always the same, there are considerable similarities and parallels among them. A survey by Halpern [12] describes some of the common threads in the area of knowledge in various fields.

Although the notions of knowledge and belief are not formally defined in [15], a number of rules related to logical consistency about these notions are proposed. The so-called *possible-world* model is introduced in [15, 12]. According to the model, besides the true states of affairs, there are a number of other possible states of affairs, or possible worlds. Some of these possible worlds may be indistinguishable from the true world to an agent. An agent is then said to *know* a fact ϕ if ϕ is true in all the worlds he thinks possible. For example, an agent may think that two states of the world are possible: in one it is sunny in Columbus, while in the other it is raining in

Columbus. However, in both states it is sunny in Phoenix. Thus, the agent knows that it is sunny in Phoenix, but does not know whether it is sunny in Columbus. As another example, a player in a poker game may consider possible all the deals consistent with the cards in his hand. A player knows a fact b if b is true in all the deals he thinks possible. Players may acquire additional information as the game progresses and may eliminate some of the worlds they consider possible.

The philosophical literature has concentrated on the one-agent case in order to emphasize the properties of knowledge. However, many applications of interest involve multiple agents. Then it becomes important to consider not only what an agent knows about the world, but also what he knows about what other agents know and do not know. This kind of reasoning is crucial in bargaining and economic decision making. It is also relevant in analyzing protocols in distributed computing systems, such as Byzantine agreement protocol [6],citebyzantine and coordinated attack problem [14]. In this context, the agents are the processes in the system.

A formal and precise treatment of the notion of knowledge in distributed systems is given by [3] where knowledge is defined in terms of *isomorphisms* of *computations* (with respect to a set of processes). In this context, the possible worlds correspond to system computations, and their being indistinguishable to an agent (or a set of agents) corresponds to computations being isomorphic with respect to a set of processes. It is also shown that many of the rules or axioms of modal logic introduced earlier hold in this setting. In this work, we are going to follow this model to generalize the notion of knowledge to that of belief in distributed systems.

On a somewhat similar treatment, knowledge is defined in terms of global and local *states* in [13]. According to that definition, a processor knows a fact b if b is true in every possible global state of the system consistent with its current information. The precise definition in this model is based on *scenarios* and *equivalence of scenarios* which are analogus to system computations and their being isomorphic in [3] respectively.

The formal notion of knowledge has been shown to be useful by formally proving, on the basis of this notion, various results on some important problems in distributed systems. It is shown in [3] that "common knowledge"is constant i.e., it can neither be attained nor be lost. A consequence of this result is that coordinated attack problem cannot be solved. It is also shown that it is impossible for a process to detect the failure of another one. Moreoever, lower bounds on number of messages required to solve certain distribued problems (such as termination detection) are obtained.

The notion of knowledge, however, has some limitations too. A fundamental result in the application of knowledge to distributed systems is that a process can only "learn" (about predicates local to its complement) by receiving a message and conversely can only "forget" by sending a message [3]. That is, if a process set P knows something about the state of a remote process set Q, Q can only change that part of its state if P sends a message. Since what P knows must be true (a property of knowledge), Q has lost some autonomy in the control of its local state. This loss of autonomy, however, may be too strict a requirement for some distributed applications (*e.g.*, ad hoc networks) or some aspects of local state (*e.g.*, failure modes which are controlled by the environment).

In this paper, we show that it is possible to generalize the notion of knowledge into one of "belief" in distributed systems. Following the model described in [3], we base this formalism on isomorphisms of system computations. We first define the notion of belief formally in such a way that knowledge is a special case of belief. Although Gettier [9] questions the validity of treating knowledge as a special case of belief, his philosophical critisim does not apply in our setting of precise mathematical formulations. We then investigate some similarities and surprising differences between the two concepts. We show that the new notion is more general and more flexible, making it more suitable to characterize formal semantics of certain distributed applications. We give examples of this in the chapter 4 by examining two common constructs in distributed systems: directory services and asynchronous leases. Both constructs are more suitably characterized in terms of belief rather than knowledge.

CHAPTER 2

MODEL OF A DISTRIBUTED SYSTEM, ISOMORPHISM AND KNOWLEDGE

In this chapter, we describe, in detail, the model of a distributed system that will be the basis for this work. This is essentially the model of computation introduced in [3] on the basis of which the formal notion of knowledge is defined.

2.1 Model of a Distributed System

A process is characterized by a set of process computations which are finite sequences of events on that process. Process computations are prefix closed. An event on a process is either a *send*, a *receive* or an *internal event*. A *send* event on a process corresponds to sending a message to another process. A *receive* event on a process corresponds to receiving a message by that process. There is no external communication associated with an *internal* event. For a set of processes P, a send event by P is a send event by some component process of P to a process outside P; similarly a receive event by P denotes receipt by some process in P of a message sent from outside P. Communications among processes in P are internal events of P. For an event e and a process set P, "e is on P" is used to denote that e is an event on some process in P. We rule out processes which have no event in any computation.

We assume that all events and all messages are distinguished; for instance, multiple occurrences of the same message are distinguished by affixing sequence numbers to them.

Let z be any sequence of events on component processes of a distributed system. The projection of z on a component process p, denoted by z_p, is the subsequence of z consisting of all events on p. A finite sequence of events z is a system computation of a distributed system if

i) for all processes p, z_p is a process computation of p

and

ii) for every receive event in z, say receipt of a message m by process a p, there is a send event, of sending m to p, which occurs earlier (i.e. in a proper prefix) than the receive in z: this event will be called the send event corresponding to the receive.

Like process computations, the set of system computations is also prefix closed.The second condition above means there are no orphan messages. Note that every receive must have a corresponding send but a send need not to have a corresponding receive.

2.2 Notation

Here are some notations we are going to use throughout the thesis. There is also a list of notations we use in the thesis in Appendix B.

We use x, y, z to denote computations; p, q for processes; P, Q for process sets; and b predicates. These symbols may be used with subscripts or superscripts. The set of all system computations will be denoted by X and the set of all processes in the system will be denoted by D. For a process set P, $\bar{P} = D - P$.

The concatenation of two sequences y and z will be denoted by $(y; z)$. For sequences y and z, $y \leq z$ denotes that y is a prefix of z; in that case (y, z) denotes the suffix of z obtained by removing the prefix y from z. The empty computation will be denoted by ϵ.

For a predicate b defined on system computations, b **at** x will be its value at computation x. We assume that all predicates are total, i.e., for every x, b **at** x is either true or false. A predicate c is constant means $c\,\mathbf{at}\,x = c\,\mathbf{at}\,y$ for all computations x and y. For a computation x, $|x|$ will denote its length and for a set A, $|A|$ will stand for its cardinality.

2.3 Isomorphism

A fundamental notion in our model of a distributed system is *isomorphism*. The notion of knowledge is formally defined in [3] in terms of isomorhism, and our formal definition of belief will also be based on this notion.

For a process p, we define a relation $[p]$ on the set of all system computations X as follows.

Definition 2.1. *For $x, y \in X$, $x[p]y$ if $x_p = y_p$.*

In words, two computations x and y are isomorphic with respect to p if p's computation is the same in x and y. For a process set P, the relation $[P]$ defined as

Definition 2.2. *$x[P]y$ if for all $p \in P$, $x[p]y$.*

Thus, $x[P]y$ means that the computations x and y cannot be distinguished by any of the processes in P. Intuitively, this corresponds to x and y being indistinguishable to P in the possible-world model.

The relation $[P]$ is an equivalence relation on X i.e., it is reflexive, symmetric and transitive. We shall denote the equivalence class of $[P]$ containing x by $[P]_x$, thus $[P]_x = \{y \in X : y[P]x\}$. The subset of $[P]_x$ where a given predicate b is true will be denoted by $[P_b]_x$, i.e., $[P_b]_x = \{y \in [P]_x : b$ **at** y is true $\}$.

It is convenient to represent all such isomorphism relations by an *isomorphism diagram:* an undirected graph whose vertices are computations and there is an edge labeled $[P]$ between vertices x and y if $[P]$ is the largest set of processes for which $x[P]y$. Observe that every vertex has a self loop labeled $[D]$. If $x[D]y$ and $x \neq y$ then y is a permutation of x.

Example 2.1.

Consider a system with two processes p and q, for which part of the isomorphism diagram, showing the relationships among four system computations, is given in Figure 2.1.

From the diagram, $x[p]y$ but not $x[q]y$. Computations x and z have the same projections for both p and q; hence one is a permutation of the other. There is no direct relationship between y and w; neither $y[p]w$ nor $y[q]w$ holds. However, there is the following indirect relationship between them: $y[p]z$ and $z[q]w$. This motivates the following definition, which is also needed in describing certain types of knowledge in the next section.

Definition 2.3. *Let $n > 0$ and P_i be process sets for $0 \leq i \leq n$.*
$x[P_0 P_1 \ldots P_n]z$ means there exists $y \in X$ such that $x[P_0 P_1 \ldots P_{n-1}]y$ and $y[P_n]z$.

We have $[PQ] = [P] \circ [Q]$, where $\circ$ is the relational composition operator. This operator is associative. In terms of isomorphism diagram, $x[P_0 P_1 \ldots P_n]z$ means there

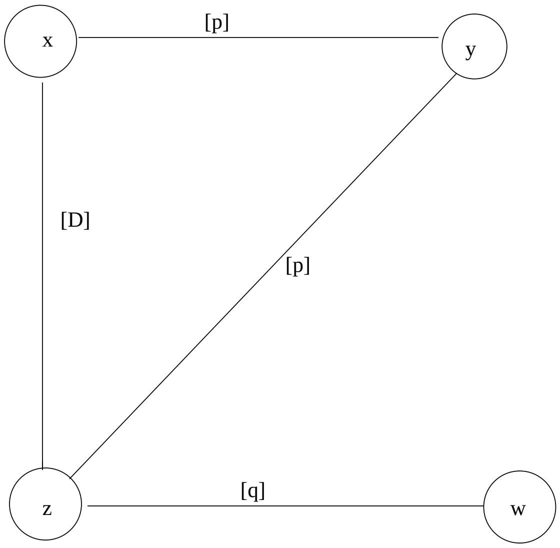

Figure 2.1: An isomorphism graph

is a path from x to z whose edges are labeled with $Q_0, Q_1, \ldots Q_n$ such that $P_i \subseteq Q_i$ for all i.

Principle of computation extension:

Let e be an event on P.

1. If e is an internal or send event then

   $(x[P]y$ and $(x; e)$ is a computation$) \Rightarrow ((y; e)$ is a computation.)

2. If e is an internal or receive event then

   $((x; e)[P]y) \Rightarrow ((y - e)$ is a computation) where $(y - e)$ is the sequence obtained by deleting e from y.

2.3.1 Process Chains

A relationship between process chains and isomorphisms is shown in [3] as described below.

Definition 2.4. *For events $e, e^{'}$ in a computation z, $e \overset{z}{\to} e^{'}$ means:*

1. $e^{'}$ is a receive and e is the corresponding send,

or

2. events e, $e^{'}$ are in the same process computation and ($e = e^{'}$ or e occurs earlier than $e^{'}$),

or

3. there exists an event $e^{''}$ such that $e \overset{z}{\to} e^{''}$ and $e^{''} \overset{z}{\to} e^{'}$.

For brevity we sometimes write $e \to e^{'}$ when the computation z is understood from the context. Observe that $e \to e$ for any event e in z.

Definition 2.5. *A computation z has a process chain $\langle P_0 P_1 \ldots P_n \rangle$ means there exist events $e_0, e_1, \ldots, e_n$, not necessarily distinct in z such that e_i is on P_i, for all $0 \leq i \leq n$, and $e_0 \to e_1 \to \cdots \to e_n$.*

For instance, if there is a chain of messages $s_0, r_1, s_1, r_2, \ldots, r_n$ in a computation z where s_i is a send by P_i and r_j is a receive by P_j for $0 \leq i, j \leq n$ such that $r_i \to s_i$ for all i and $s_k \to s_l$ for all $k < l$, then there is a process chain $\langle P_0 P_1 \ldots P_n \rangle$ in z. We observe that any occurrence of P in a process chain may be replaced by PP, or vice versa. Also, if x is a subsequence of a computation y such that for every e in x the following holds: every e', where $e' \overset{y}{\to} e$, is also in x, and $e' \overset{x}{\to} e$. Then x is a computation.

The following theorem is called the fundamental theorem of process chains.

Theorem 2.1. *Let z be a computation and $x \leq z$. Let $P_1, P_2, \ldots, P_n$ be process sets. Then $x[P_1 P_2 \ldots P_n]z$ or there is a process chain $\langle P_1 P_2 \ldots P_n \rangle$ in (x, z).*

2.4 Knowledge

The notion of knowledge can be defined in terms of isomorphisms [3] and it can be shown that with this definition, the "knowledge axioms" [15, 14] are satisfied.

Definition 2.6. $(P$ **knows** $b)$ **at** x *if for all* $y \in [P]_x, b$ **at** y.

Symbolically, P **knows** b **at** x means $[P]_x = [P_b]_x$. Intuitively, this means in all the worlds $[P]_x$ the agent P thinks possible, the predicate b is true.

Note that b may itself be a predicate of the form Q **knows** b' for some predicate b' in the above definition. Below are some facts about knowledge predicates. Throughout this thesis, letters x, y, z will denote computations, P, Q will denote process sets and b, b' will denote predicates. A number of facts about knowledge predicates can be derived directly from the definition [3].

Lemma 2.1.

1. P **knows** b **at** x = *for all* $y \in [P]_x$, P **knows** b **at** y
2. $y \in [P]_x \Rightarrow (P$ **knows** b **at** $x = P$ **knows** b **at** $y)$
3. P **knows** $b \Rightarrow P \cup Q$ **knows** b
4. P **knows** $b \Rightarrow b$
5. P **knows** $b \vee \sim (P$ **knows** $b)$
6. $(P$ **knows** $b) \wedge (P$ **knows** $b') \Rightarrow P$ **knows** $(b \wedge b')$
7. $(P$ **knows** $b) \vee (P$ **knows** $b') \Rightarrow P$ **knows** $(b \vee b')$
8. $(P$**knows** $\sim b) \Rightarrow \sim (P$ **knows** $b)$

9. $(P$ **knows** $b) \wedge (b \Rightarrow b^{'}) \Rightarrow P$ **knows** $b^{'}$

10. P **knows** $(P$ **knows** $b) = P$ **knows** b

11. P **knows** $(\sim (P$ **knows** $b)) = \sim (P$ **knows** $b)$

12. P **knows** $c \vee P$**knows** $\sim c$*, for any constant* c*.*

We comment on some of the properties above. Number 4 says that only true things are known, a property which is universally agreed to be true. We will argue in the next chapter that this property actually makes the notion of knowledge inappropriate (too strong) to apply in certain situations. The validity of 11 is questioned on philosophical grounds [14].

There are situations where multiple levels of knowledge such as, p**knows** (q**knows** b) **at** x are useful. The relation defined in Definition 2.3 turns out to be useful to describe this type of knowledge because

p **knows** (q **knows** b) **at** x

=for all $y : x[p]y : q$ **knows** b **at** y

=for all $y : x[p]y :$ (for all $z : y[q]z : b$ **at** z)

=for all $z : x[pq]z : b$ **at** z

In the next section, we describe an important class of predicates: local predicates.

2.4.1 Local Predicates

Local predicates are important in terms of knowledge transfer. A predicate is local to a process p if its value can only be changed by events on p. The following definitions formalize this notion. Let b be a predicate on system computations, and P be a set of processes. We define a predicate P **sure** b as follows.

Definition 2.7. $(P$ **sure** $b)$ **at** x *if* $(P$ **knows** $b)$ **at** x *or* $(P$ **knows** $\sim b)$ **at** x.

In other words, $(P$ **sure** $b)$ **at** x means that P knows the value of b at x.

We note that P **sure** b corresponds to "P knows whether b" in [15].

Definition 2.8. *b is local to P if, for all x,* $(P$ **sure** $b)$ **at** x.

That is, the value of b is always known to P. Local predicates capture our intuitive notion of a predicate whose value is controlled by the actions of processes to which it is local.

We have the following facts about local predicates.

Lemma 2.2.

1. *b is local to P* $\wedge\ y \in [P]_x \Rightarrow (b$ **at** $x = b$ **at** $y)$
2. *b is local to P* $\Rightarrow b = P$ **knows** b
3. *b is local to P* $= (\sim b)$ *is local to P*
4. *b is local to P* $\Rightarrow Q$ **knows** $b = Q$ **knows** $(P$ **knows** $b)$
5. $(P$ **knows** $b)$ *is local to P*
6. *b is local to P, b is local to Q, and P, Q are disjoint* $\Rightarrow$ *b is a constant*
7. *b is a constant* $\Rightarrow$ *b is local to P*
8. $(P$ **sure** $b)$ *is local to P*

2.4.2 Transfer of Knowledge

The following results about transfer of knowledge are proved in [3].

Theorem 2.2. *For arbitrary process sets $P_1, \ldots, P_n$, $n \geq 1$, predicate b and computations x, y,*

$(P_1$ **knows** $\ldots P_n$ **knows** b **at** x *and* $x[P_1 \ldots P_n]y) \Rightarrow (P_n$ **knows** b **at** $y)$

The following lemma from [3] is of fundamental importance in transfer of knowledge. We state it below and give an alternative and more elementary proof.

Lemma 2.3. *Let b be a predicate which is local to $\bar{P}$ and $(x; e)$ a computation where e is an event on P. Then*

i) If e is a receive then

$(P$ **knows** b **at** $x) \Rightarrow (P$ **knows** b **at** $(x; e))$ *(Knowledge is not lost).*

ii) If e is a send then

$(P$ **knows** b **at** $(x; e)) \Rightarrow (P$ **knows** b **at** $x)$ *(Knowledge is not gained).*

iii) If e is an internal event, then

$(P$ **knows** b **at** $x) = (P$ **knows** b **at** $(x; e))$ *(Knowledge is unchanged).*

Proof. We prove i). Let z be a computation such that $(x; e)[P]z$. Consider $(z - e)$ which is a computation by the principle of computation extension. We have $x[P](z - e)$, therefore b at $(z - e)$ and b at z (since b is local to $\bar{P}$). $\square$

The above lemma has the following important corollaries.

Corollary 2.1. *(b is local to $\bar{P}$, $\sim$ $(P$ **knows** b **at** $x)$, P **knows** b **at** $y, x \leq y) \Rightarrow$ (P receives a message in (x,y))*

Corollary 2.2. *(b is local to* $\bar{P}, P$ **knows** b **at** $x, \sim (P$ **knows** b **at** $y), x \leq y) \Rightarrow$ *(P sends a message in (x,y))*

The following two theorems in [3] shows how a chain of knowledge is gained or lost.

Theorem 2.3. *(How knowledge is gained) Let x,y be computations where* $x \leq y$*,* $\sim (P_n$ **knows** b **at** $x)$ *and* $(P_1$ **knows** $\ldots P_n$ **knows** b **at** $y)$*, for some process sets* $P_1, \ldots, P_n$*. Then there is a process chain* $\langle P_1 \ldots P_n \rangle$ *in (x,y). Furthermore, if b is local to* $\bar{P}_n$*, then* P_n *has receive event in (x,y) such that* b **at** z *holds for every prefix z of y which includes the corresponding send event.*

Theorem 2.4. *(How knowledge is lost) Let x,y be computations where* $x \leq y$*,* $(P_1$ **knows** $\ldots P_n$ **knows** b **at** $x) \sim (P_n$ **knows** b **at** $y)$ *, for some process sets* $P_1, \ldots, P_n$*. Then there is a process chain* $\langle P_1 \ldots P_n \rangle$ *in (x,y). Furthermore, if b is local to* $\bar{P}_n$*, then* P_n *has a send event in (x,y).*

2.4.3 Applications of Knowledge

Now we briefly mention some of the applications of the formal notion of knowledge in distributed systems. The following results are proven in [3] and we refer the reader to that paper for details.

- In a system with more than one process, for any predicate b, " b is a common knowledge" is constant, i.e., common knowledge can neither be gained nor lost in distributed systems. The predicate *b is a common knowledge* is defined as the greatest fixed point of the equation

  b is common knowledge $=$ *b and* $(p$ **knows** $b)$ *is common knowledge*

Intuitively, this means b is true, everybody knows b, everybody knows that everybody knows b etc.

- It is shown that $\bar{P}$ **knows** $(P$ **unsure** $f)$ **at** x for all $x \in X$ where f is the local predicate of $\bar{P}$ denoting that $\bar{P}$ has failed. This result shows that it is impossible for a process to detect failure of another process.

- In every computation of a solution to the mutual exclusion problem, there is a process chain $\langle p_1 \ldots p_n \rangle$ where p_i is the ith process to enter its critical section. On the bases of this result, they show that a solution to a version of the distributed dining philosophers problem [2] requires no more than twice the number of messages in an optimal scheme.

- Any algorithm which detects termination of an underlying computation requires at least as many overhead messages, in general, for detection as there are messages in the underlying computation.

CHAPTER 3

BELIEF IN DISTRIBUTED SYSTEMS

3.1 Motivation

As explained in more detail in the next chapter, the notion of knowledge is too restrictive in terms of describing semantics of certain distributed applications where processes are loosely coupled. A process's knowledge about predicates local to its complement cannot be lost by a receive of a message or cannot be gained by a send of a message. This may turn out to be a too strong of a restriction of knowledge. For example, consider a system consisting of a printer and a number of processes. Suppose we have the predicate b: "There is paper in paper tray" and this predicate is local to the printer. Also assume that all the processes in the system come to "know" b (by receiving a message from the printer for example). Then as long as any of the processes knows b, it must remain true and it cannot be falsified unless every process who knows b sends a message to the printer. Until that happens, printer must maintain paper in the paper tray. Obviously, this is not how this situation should be characterized. So the notion of knowledge is not appropriate to describe this situation.

On the other hand, even if a predicate b holds for the "majority" (even if all but one) of computations that are isomorphic to a given computation x with respect to

P, P still does not know b at x. But we still want to be able to say something about P "knowing" b at x in that case. In response to these issues, we define the notion of "belief" in distributed systems which is a more general and flexible notion.

3.2 Formal Definition

Recall that P **knows** b **at** x was defined as $[P]_x = [P_b]_x$. A natural way (and suggested by [3]) to generalize knowledge to belief seems to be the following: process P believes b at x means b holds for most (or some) computations isomorphic to x with respect to P. More precisely, it seems like we need to quantify belief by attaching a real number α with $0 \leq \alpha \leq 1$. So a first attempt to define P $\mathbf{bel}_\alpha$ b **at** x, read "P believes b at x with α", would be

$$\frac{|[P_b]_x|}{|[P]_x|} \geq \alpha$$

However, there is a technical problem with this definition. Since the computations can be arbitrarily long, the set $[P]_x$ can be infinite, so can the numerator of the above expression. (If the numerator is finite and the denominator is infinite, then the value of the fraction would be 0). Therefore, we need to define the belief more carefully. To this end, we first introduce some notation.

For a natural number N, and a computation x, let $X_N = \{x \in X : |x| \leq N\}$, i.e., X_N is the set of all system computations of length at most N. If there are n processes in the system and each process can take one of the k possible actions, then $|X_N| \leq (kn)^N$. For each computation x, process set P, predicate b and natural number N such that $N \geq |x_p|$, we define a rational number $a_N = a_N(P, b, x)$

$$a_N = \frac{|X_N \cap [P_b]_x|}{|X_N \cap [P]_x|}$$

Associated to each $a_N(P, b, x)$ is

$$\tilde{a}_N = a_N(P, \sim b, x) = \frac{|X_N \cap [P_{\sim b}]_x|}{|X_N \cap [P]_x|}$$

So, we obtain two sequences of rational numbers $(a_N)_{N \geq 0}$, $(\tilde{a}_N)_{N \geq 0}$ with $0 \leq a_N, \tilde{a}_N \leq 1$ and $a_N + \tilde{a}_N = 1$ for all N. Note that a_N represents the fraction of the computations of length at most N in the class $[P]_x$ for which b is true. A second attempt to define P $\mathbf{bel}_\alpha$ a_N would be $\lim\limits_{N \to \infty} a_N \geq \alpha$. Unfortunately, there is still a technical problem with this; not every sequence of real numbers has a limit. However, every sequence of real numbers has a limit infimum (Appendix A) and that is what we use.

Definition 3.1. *P $\mathbf{bel}_\alpha$ b **at** x if and only if*

$$\liminf_{N \to \infty} a_N(P, b, x) \geq \alpha \tag{3.1}$$

Although $\lim\limits_{N \to \infty} a_N$ need not exist for an arbitrary sequence a_N, $\liminf\limits_{N \to \infty} a_N$ always does. When $\lim\limits_{N \to \infty} a_N$ does exist, it is equal to $\liminf\limits_{N \to \infty} a_N$. Moreover, since $0 \leq a_N \leq 1$ for all N, $0 \leq \liminf\limits_{N \to \infty} a_N \leq 1$. Also note that $\liminf\limits_{N \to \infty} a_N$ is the largest number β such that P $\mathbf{bel}_\beta$ b **at** x.

Notation: We will sometimes use the notation $\liminf a_N$ to mean $\liminf\limits_{N \to \infty} a_N$ and $\lim a_N$ to mean $\lim\limits_{N \to \infty} a_N$.

Some of the properties of limit infimum and limit supremum are as follows (see Appendix A for details):

- $\liminf(a_n) + \liminf(b_n) \leq \liminf(a_n + b_n) \leq \limsup(a_n + b_n) \leq \limsup(a_n) + \limsup(b_n)$

- If $\lim(a_n) = a$ then $\liminf(a_n + b_n) = a + \liminf(b_n)$ and $\limsup(a_n + b_n) = a + \limsup(b_n)$

3.3 Properties of Belief

Our first observation is that $P\,\mathbf{knows}\,b\,\mathbf{at}\,x$ implies $P\,\mathbf{bel}_\alpha\,b\,\mathbf{at}\,x$ for any $0 \leq \alpha \leq 1$. Conversely, when the sequence a_N turns out to be the constant sequence 1, then we can take $\alpha = 1$ and in this case $P\ \mathbf{bel}_1\ b\ \mathbf{at}\ x$ is the same as $P\,\mathbf{knows}\,b\,\mathbf{at}\,x$. However, $\liminf a_N$ being 1 (a belief of 1) is not necessarily the same as P **knows** b **at** x. The following is an example of this situation.

Example 3.1.

Consider the following rather trivial system consisting of two processes P and Q. Let P and Q's computations consist of s_p^* and s_q^* respectively, where s_p and s_q are skip events on the corresponding processes. Let x be $s_q s_q$ and let b be the predicate "There are at least 2 events". Then it is not the case that P **knows** b **at** x, since $y[P]x$ where $y = s_q$, yet $\sim b$ **at** x is true. However, we have

$$P\ \mathbf{bel}_1\ b\ \mathbf{at}\ x.$$

In fact, $a_N = \frac{N-1}{N+1}$ for $N \geq 1$.

Below are more properties of belief.

Lemma 3.1.

1. *If* $\beta > \liminf a_N$ *then* $\sim (P\ \mathbf{bel}_\beta\ b\ \mathbf{at}\ x)$

2. $P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x$ *implies* $P\ \mathbf{bel}_\beta\ b\ \mathbf{at}\ x$ *for all* $\beta \leq \alpha$

3. $(P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x) \vee \sim (P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x)$

4. P $\mathbf{bel}_0$ b *for any predicate* b.

5. $(a_N = 1, \forall N)$ *implies* $(P\ \mathbf{bel}_1\ b\ \mathbf{at}\ x) \Leftrightarrow (P\ \mathbf{knows}\ b\ \mathbf{at}\ x)$

6. *If there exists an integer* N_0 *such that* $|x| \leq N_0$ *for all* $x \in X$ *then*

   i) $a_N = a_M$ *for all* $N, M \geq N_0$*, that is the sequence* a_N *eventually becomes constant.*

   ii) $(P\ \mathbf{bel}_1\ b\ \mathbf{at}\ x) \Leftrightarrow (P\ \mathbf{knows}\ b\ \mathbf{at}\ x)$

7. *If* $x[P]y$ *then* $(P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x) \Leftrightarrow (P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ y)$

8. $(P\ \mathbf{bel}_\alpha\ b) \wedge (b \Rightarrow b') \Rightarrow (P\ \mathbf{bel}_\alpha\ b')$

9. $(P\ \mathbf{bel}_\alpha\ b) \wedge (P\ \mathbf{bel}_\beta\ b') \Rightarrow P\ \mathbf{bel}_{\max\{0,\alpha+\beta-1\}}\ (b \wedge b')$

10. $((P\ \mathbf{bel}_\alpha\ b) \wedge (P\ \mathbf{bel}_\beta\ b')) \Rightarrow P\ \mathbf{bel}_{\max\{\alpha,\beta\}}\ (b \vee b')$

11. $((P\ \mathbf{bel}_\alpha\ b) \vee (P\ \mathbf{bel}_\beta\ b')) \Rightarrow P\ \mathbf{bel}_{\min\{\alpha,\beta\}}\ (b \vee b')$

12. $(P\mathbf{bel}_\beta \sim b\ \mathbf{at}\ x) \Rightarrow \beta \leq 1 - \liminf a_N(P, b, x)$. *Moreover, if* $\lim_{N\to\infty} a_N$ *exists and is equal to* α *then*

    $(P\ \mathbf{bel}_\alpha\ b) \Leftrightarrow (P\mathbf{bel}_{1-\alpha} \sim b)$

13. $P\ \mathbf{bel}_\alpha\ b \Leftrightarrow P\ \mathbf{knows}\ (P\ \mathbf{bel}_\alpha\ b)$

14. $P\ \mathbf{bel}_\alpha\ b$ *is local to* P.

15. $P\ \mathbf{bel}_\alpha\ (Q\ \mathbf{knows}\ b) \Rightarrow P\ \mathbf{bel}_\alpha\ b$

16. *If* $\alpha > 0$ *then* $P\ \mathbf{bel}_\alpha\ (P\ \mathbf{bel}_\beta\ b) \Rightarrow P\ \mathbf{bel}_\beta\ b$

17. $([P \cup Q]_x \subseteq [P_b]_x \cup [Q_b]_x) \Rightarrow P \cup Q\ \mathbf{knows}\ b\ \mathbf{at}\ x$

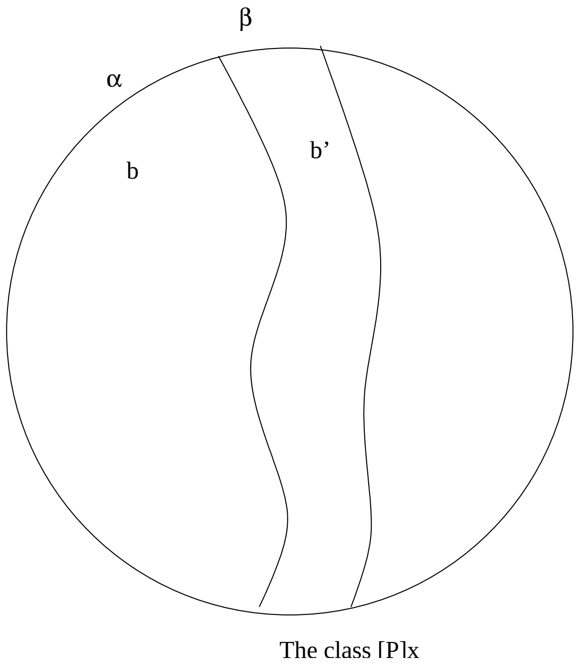

Figure 3.1: Venn Diagram for Belief Property 10

18. $(P\,\mathbf{knows} \sim b$ **at** $x) \Rightarrow \sim (P$ $\mathbf{bel}_\alpha$ b **at** $x)$ *for any* $\alpha > 0$.

Proof. 1-5 follow immediately from the definition.

6i: If the lengths of the computations are bounded by N_0 then $|X| = |X_{N_0}| = |X_M|$ for any $M \geq N_0$. Therefore, $a_N = a_M$ for $N, M \geq N_0$. 6ii: In this case if $\liminf a_N = 1$ then the sequence a_N has to be the constant sequence 1, since $X_M \subseteq X_N$ for $M \leq N$.

7:Follows from the fact that if x and y are in the same equivalence class with respect to P, then $[P]_x = [P]_y$ and $[P_b]_x = [P_b]_y$.

8: When $b \Rightarrow b'$, $[P_b]_x \subseteq [P_{b'}]_x$.

Proofs of 9-11 can be illustrated by a diagram, such as one in Figure 3.1.

12: Since $1 = a_N + \tilde{a}_N$ for all N, $1 = \liminf(a_N + \tilde{a}_N) \geq \liminf(a_N) + \liminf(\tilde{a}_N)$. So, $\liminf(\tilde{a}_N) \leq 1 - \liminf(a_N)$ and $\liminf(\tilde{a}_N)$ is the largest number γ such that $P\,\mathbf{bel}_\gamma \sim b$. If $\liminf(a_N) = \lim(a_N) = \alpha$, then $1 = \liminf(a_N + \tilde{a}_N) = \lim(a_N) + \liminf(\tilde{a}_N) = \alpha + \liminf(\tilde{a}_N)$, equivalently, $\liminf(\tilde{a}_N) = 1 - \alpha$.

13: Let P $\mathbf{bel}_\alpha$ b **at** x and suppose $y[P]x$. Then by 7, P $\mathbf{bel}_\alpha$ b **at** y, hence P **knows** (P $\mathbf{bel}_\alpha$ b) **at** x. The other direction is a known property of knowledge.

14: By 13 we have P **knows** $(P\ \mathbf{bel}_\alpha\ b) \vee P$ **knows** $(\sim (P\ \mathbf{bel}_\alpha\ b) = P\ \mathbf{bel}_\alpha\ b \vee \sim (P\ \mathbf{bel}_\alpha\ b)$=true.

15: Since Q **knows** $b \Rightarrow b$, this property follows from 8.

16: If $\alpha > 0$ and P $\mathbf{bel}_\alpha$ (P $\mathbf{bel}_\beta$ b) **at** x, then there exists a computation $y \in [P]_x$ such that P $\mathbf{bel}_\beta$ b **at** y. Since x and y are in the same equivalence class with respect to P, P $\mathbf{bel}_\beta$ b **at** x as well.

17: Let $y \in [P \cup Q]_x$. Then $y \in [P_b]_x$ or $y \in [Q_b]_x$. Either case b **at** y is true, therefore $P \cup Q$ **knows** b **at** x.

18: If P**knows** $\sim b$ **at** x then $a_N(P, b, x) = 0$ for all N. Therefore, $\liminf a_N = \lim a_N = 0$. □

We note that 13 ($\Rightarrow$ direction) is taken as an axiom in [10]. Hintikka argues in [15] that property 16 is not valid (though his notion of belief does not involve an analogue of the quantifier α).

Notation: By property 2 above, whenever P believes b at x with α, P also believes b at x with any $\beta \leq \alpha$ and $\liminf a_N$ is the largest number α such that P believes b at x with α. We shall denote this maximal α by α_M.

Now we present an example that shows the inequality in 14 can be strict. In other words we have,

Observation I: $(P\ \mathbf{bel}_{\alpha_M}\ b\ \mathbf{at}\ x) \not\Rightarrow (P\mathbf{bel}_{1-\alpha_M} \sim b\ \mathbf{at}\ x)$

Example 3.2.

Let a system consist of two processes P and Q where P's computations consist of s_p^* and Q's computations consist of $(h|t)^*$. Let x be the empty computation ϵ and b be the predicate "There are an odd number of events". Then we compute that

$$a_N(P, b, x) = \begin{cases} \frac{\frac{2^{N+2}-2}{3}}{2^{N+1}-1} = \frac{2}{3} & \text{if } N \text{ is odd} \\ \\ \frac{\frac{2^{N+1}-2}{3}}{2^{N+1}-1} & \text{if } N \text{ is even} \end{cases}$$

Therefore, $\liminf a_N = \frac{1}{3}$ and $(P\ \mathbf{bel}_{\frac{1}{3}}\ b\ \mathbf{at}\ x)$ (and $\sim (P\ \mathbf{bel}_{\alpha}\ b\ \mathbf{at}\ x)$ for any $\alpha > \frac{1}{3}$).

Similarly,

$$\tilde{a}_N = \begin{cases} \frac{\frac{2^{N+1}-1}{3}}{2^{N+1}-1} = \frac{1}{3} & \text{if } N \text{ is odd} \\ \\ \frac{\frac{2^{N+2}-1}{3}}{2^{N+1}-1} & \text{if } N \text{ is even} \end{cases}$$

Therefore, $\liminf \tilde{a}_N = \frac{1}{3}$, hence $(P\mathbf{bel}_{\frac{1}{3}} \sim b\ \mathbf{at}\ x)$ (and $\sim (P\mathbf{bel}_{\beta} \sim b\ \mathbf{at}\ x)$ for any $\beta > \frac{1}{3}$).

Example 3.3.

If we modify the previous example so that Q's computations are $(h_1|h_2|\ldots|h_k)^*$ and everything else is the same, then we obtain the rather surprising (and disturbing!) result that $P\ \mathbf{bel}_{\frac{1}{k+1}}\ b\ \mathbf{at}\ x$ and $P\mathbf{bel}_{\frac{1}{k+1}} \sim b\ \mathbf{at}\ x$. In the next section, we are going to propose an alternative definition of belief according to which both of these beliefs will be $\frac{1}{2}$.

The following lemma is proven using Theorem 4 in [3].

Lemma 3.2. *For arbitrary process sets $P_1 \ldots P_n$, predicate b, and computations x,y, if $(P_1\ \mathbf{knows} \ldots P_{n-1}\ \mathbf{knows}\ (P_n\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x))$ and $x[P_1 \ldots P_n]y$ then $(P_n\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ y)$.*

Proof. Assume that $(P_1\ \mathbf{knows} \ldots P_{n-1}\ \mathbf{knows}\ (P_n\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x))$ and $x[P_1 \ldots P_n]y$. We need to show $(P_n\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ y)$.

Since $x[P_1 \ldots P_n]y$, there exists a computation z such that $x[P_1 \ldots P_{n-1}]z$ and $z[P_n]y$. By Theorem 4 in [3], we have $P_{n-1}\ \mathbf{knows}\ (P_n\ \mathbf{bel}_\alpha\ b)\ \mathbf{at}\ z$ which implies $P_n\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ z$. Finally, since $z[P_n]y$, $P_n\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ y$. □

3.4 Transfer of Belief

The Lemma 2.3 is of fundamental importance in transfer of knowledge. We show that the corresponding facts of i) and ii) do not hold for belief. However, iii) does.

Lemma 3.3. *Let b be a predicate that is local to $\bar{P}$ and $(x; e)$ a computation where e is an internal event on P. Then*

$$(P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x) = (P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ (x; e))$$

Proof. We will show $a_N(P, b, x) = a_N(P, b, (x; e))$. Let $y \in X_{N+|(x;e)|} \cap [P]_{(x;e)}$. So, $|y| = N + |x| + 1$ and $y[P](x; e)$. Consider $(y - e)$, which is a computation by the principle of computation extension. We have $(y - e)[P]x$ and $|(y - e)| = N + |x|$, therefore, $(y - e) \in X_{N+|x|} \cap [P]_x$. Hence, $|X_{N+|(x;e)|} \cap [P]_{(x;e)}| \leq |X_{N+|x|} \cap [P]_x|$.

Conversely, let $y \in X_{N+|x|} \cap [P]_x$. Then, $|y| = N + |x|$ and $y[P]x$. Consider $(y; e)$, which is a computation by the principle of computation extension. We have $|(y; e)| = |y| + 1 = N + |x| + 1 = N + |(x; e)|$ and $(y; e)[P](x; e)$. Therefore, $(y; e) \in X_{N+|(x;e)|} \cap [P]_{(x;e)}$ and so $|X_{N+|(x;e)|} \cap [P]_{(x;e)}| \geq |X_{N+|x|} \cap [P]_x|$. Consequently, $|X_{N+|(x;e)|} \cap [P]_{(x;e)}| = |X_{N+|x|} \cap [P]_x|$.

Similarly, using the facts

b **at** $y \Leftrightarrow b$ **at** $(y - e)$

b **at** $y \Leftrightarrow b$ **at** $(y; e)$

we show

$|X_{N+|(x;e)|} \cap [P_b]_{(x;e)}| = |X_{N+|x|} \cap [P_b]_x|.$

$\square$

Corollary 3.1. *Let b be local to $\bar{P}$. If $\sim (P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x)$ and $(P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ y)$, $x \leq y$, then P sends or receives a message in (x, y).*

Corollary 3.2. *Let b be local to $\bar{P}$. If $(P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x)$, $\sim (P\ \mathbf{bel}_\beta\ b\ \mathbf{at}\ x)$ for any $\beta > \alpha$ and $(P\ \mathbf{bel}_\beta\ b\ \mathbf{at}\ y)$, $x \leq y$, then P sends or receives a message in (x, y).*

Unlike knowledge, a process's belief about predicates local to its complement may decrease by a reception of a message and may increase by a send of a message. The examples in the next section illustrate these situations.

3.5 Examples

Example 3.4.

This example shows that belief can go down by a receipt of a message. Let a system consist of two processes P and Q. Let the computations of Q be one of the following two types:

$$hs_q^* S_h s_q^*$$

and

$$ts_q^* S_t s_q^*.$$

A computation of P is either of the following forms:

$$s_p^* R_h$$

$$s_p^* R_t$$

where s_q and s_p are skips on Q and P respectively, S_h and R_h are the events of send of h by Q and receipt of h by P respectively. (Similarly for S_t and R_t) Let x be the computation hS_h and b be the predicate " an h occurred ". Then we have

$$P \ \mathbf{bel}_{\frac{1}{2}} \ b \ \mathbf{at} \ x.$$

Here, $a_N(P, b, x) = \frac{\frac{N(N+1)}{2}}{N(N+1)+1}$.

If we take e to be R_h then we have

$$P \ \mathbf{bel}_1 \ b \ \mathbf{at} \ (x; e).$$

In fact we have P **knows** b **at** $(x; e)$.

Now let y be tS_t and e be R_t. With the same predicate b, we have the following:

$$P \ \mathbf{bel}_{\frac{1}{2}} \ b \ \mathbf{at} \ y$$

and

$$P \ \mathbf{bel}_0 \ b \ \mathbf{at} \ (y; e).$$

This cannot happen if belief is replaced with knowledge.

Example 3.5.

This example shows that belief can go up by a send.

Using the notation and the set up of the previous example, let P's computations be $s_p^* S_h$'s or $s_p^* S_t$'s and Q's computations be $R_h h s_q^*$'s or $R_t t s_q^*$'s. Let $x = \epsilon$, e be S_h

and b be the predicate "An h occurred". Then we have

$$P \ \mathbf{bel}_0 \ b \ \mathbf{at} \ x$$

and

$$P \ \mathbf{bel}_1 \ b \ \mathbf{at} \ (x; e).$$

That is, belief strictly increases by a send of a message.

The following example shows that a belief of 1 may decrease to a non-zero belief.

Example 3.6.

Extending Example 3.4, let Q have the following additional computations

$$S_s$$

$$s_q S_s$$

$$s_q s_q S_s$$

and let P have the additional computation

$$s_p^* R_s$$

where S_s denotes the send of s (intuitively send of a stop) and R_s is the reception of s. Let b be the predicate "there are at least 3 events", x be the computation $s_q S_s$ and let e be R_s. Then we have

$$P \ \mathbf{bel}_1 \ b \ \mathbf{at} \ x$$

and

$$P \ \mathbf{bel}_{\frac{2}{3}} \ b \ \mathbf{at} \ (x; e).$$

3.6 An Alternate Definition

The results in Example 3.2 and Example 3.3 above are quite unexpected. We would not expect that the belief about the parity of number of events in a computation would depend on the number of available actions. (In fact we expect that the belief be 0.5 independent of number of choices available). This suggests that other possible definitions of belief may exist which would avoid this situation.

In a sense, defining belief in terms of lim inf is too conservative. Using lim sup, on the other hand, is too optimistic (besides it would still lead to similar problems). One possible solution is to take the average of lim inf and lim sup, i.e., we alternatively define P $\mathbf{bel}_\alpha$ b **at** x to mean

$$\frac{\liminf a_N(P,b,x) + \limsup a_N(P,b,x)}{2} \geq \alpha$$

in our usual notation.

We remark that this definition coincides with the earlier definition when $\lim a_N$ happens to exist. In this case,

$$\liminf a_N = \limsup a_N = \frac{\liminf a_N + \limsup a_N}{2} = \lim a_N$$

With this new definition, we see that most of the properties in Lemma 3.1 hold. However, property 14 in that lemma is replaced with the following.

Lemma 3.4. *With the definition introduced in this section, we have*

$$(P\ \mathbf{bel}_\alpha\ b\ \mathbf{at}\ x) \Rightarrow (P\,\mathbf{bel}_{1-\alpha} \sim b\ \mathbf{at}\ x)$$

for the maximal belief level of α *i.e., for* $\alpha = \frac{\liminf a_N(P,b,x)+\limsup a_N(P,b,x)}{2}$.

Proof. Recall that $\tilde{a}_N$ is the sequence associated with the predicate $\sim b$. We have $a_N + \tilde{a}_N = 1$. Therefore, $\tilde{a}_N = 1 - a_N$ and

$$\begin{aligned}\liminf \tilde{a}_N &= \liminf(1 - a_N) \\ &= 1 + \liminf(-a_N) \\ &= 1 - \limsup(a_N)\end{aligned}$$

Similarly,

$$\begin{aligned}\limsup \tilde{a}_N &= \limsup(1 - a_N) \\ &= 1 + \limsup(-a_N) \\ &= 1 - \liminf(a_N)\end{aligned}$$

Hence,

$$\begin{aligned}\frac{\liminf \tilde{a}_N + \limsup \tilde{a}_N}{2} &= \frac{2 - (\liminf a_N + \limsup a_N)}{2} \\ &= 1 - \frac{\liminf a_N + \limsup a_N}{2} \\ &= 1 - \alpha \quad \text{(by assumption)}\end{aligned}$$

□

A consequence of this lemma is that with this definition of belief neither of the situations illustrated in Examples 3.2 and Example 3.3 will occur. That is, **Observation I** does not hold and all the (maximal) beliefs in those two examples are $\frac{1}{2}$. This, together with the fact that other properties of belief remain valid, suggests that the alternate definition of belief is a better one.

CHAPTER 4

APPLICATIONS OF BELIEF

In this chapter, we consider two common constructs in distributed systems: directories and leases. Both constructs have proven to be very useful in the construction of real, asynchronous distributed systems, but a knowledge-based analysis does not accurately capture their utility. Instead, belief can be used to quantify the benefits of these constructs.

4.1 Distributed Directories

For bootstrapping a distributed application, components must find their remote counterparts. Typically, some form of naming service or directory is used (e.g. the CORBA naming service [20], the Java RMI registry [23], or UDDI [18]). This model is based on the assumption that such a directory is more easily found than specific application components.

An entry in a directory is often thought of as reflecting what the directory *knows* about the remote application component. For example, an entry might include what interface is used to access the component and where the component is located. This knowledge is gained when the directory receives a message (a registration) from this component.

For real, loosely-coupled, distributed systems, however, this knowledge-based model of a directory service is too stringent. The knowledge-based characterization requires that the directory can only "forget" information by sending a message. Thus, an application component that is registered with a directory service must maintain the same interfaces and be at the same location as indicated in its registry, until it receives a message from the directory! Thus, any information registered with a directory service (modeled by knowledge) represents a loss of volition on the part of the application component. This loss of volition is not appropriate for loosely-coupled systems (*e.g.*, with high mobility, or independent reversioning of interfaces).

Even in loosely-coupled systems, however, directories are still useful constructs. They are useful not because their entries represent knowledge, but rather because they represent *belief*. The remote application component may or may not have the attributes encoded in its registration, but the directory believes (with a certain threshold) that the entry is correct. Other components, by querying the directory service, would learn about this belief. For a component C using a directory D that has an entry r regarding a registered component, the property obtained is $C\,\mathbf{knows}\,D\,\mathbf{bel}_\alpha\,r$. With this model of directories, application components can still change their state, so long as this possibility is reflected in the threshold (α) of belief.

4.2 Asynchronous Leases

A lease is a time-driven mechanism for releasing remote resources. Leases have been used to address a variety of problems in distributed system design, including cache consistency [11] and garbage collection [5]. They can also be used to isolate a system from the failure of an individual component.

For example, in garbage collection, a reference is released only with an accompanying lease. The server guarantees the availability of the referenced object only for the duration of the lease. If the client wishes to maintain the validity of the reference it holds, the corresponding lease must be renewed with the server. If the client crashes, its lease eventually expires, and the server can garbage collect the referenced object.

Leases can be characterized using knowledge either from the point of view of the holder or the grantor of the lease. In the case of garbage collection, for example, granting a lease reflects knowledge that a remote reference exists. The expiration of a lease reflects the knowledge that a (valid) remote reference no longer exists.

In a purely asynchronous system, timeouts can not be reliably used. In this case, the lease mechanism still has application, but the expiration of a lease triggers a message and acknowledgment protocol between the grantor and the holder. It is the reception of an acknowledgment from the holder that allows the grantor to know that the resource represented by the lease is no longer held by the remote client.

When failures can occur, however, the grantor cannot rely on an acknowledgment to learn something about the lease holder (since the lease holder may have crashed). In these cases, the grantor must be able to learn something from its *sending* of a message. This suggests that asynchronous leases should be modeled as reflections of *belief* rather than knowledge.

For a server S granting a lease to a client C, the property obtained is $S\ \mathbf{bel}_\alpha$ C holds res (where res is the corresponding resource). The belief threshold α may be a function of the likelihood of C crashing. The "expiration" of an asynchronous lease is a local action at the server. By Lemma 3.3, however, belief can not change by a local action alone. Therefore, the expiration of a lease must correspond to the server sending a notification to the client. After the expiration, a new belief property of the form $S\,\mathbf{bel}_\beta \sim (C \text{ holds res})$ is obtained. It is this belief that justifies garbage collection by the server.

Because belief (even with a threshold of 1) is not necessarily knowledge, holding (or granting) a lease is not a guarantee of availability (or unavailability) of the corresponding resource. This belief-based formulation is exactly the best-effort semantics of real distributed leases, such as those provided in Jini [5].

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In this thesis we show that it is possible to generalize the notion of knowledge in distributed systems to that of belief in a useful way. We have proposed a formalism for the notion of belief in distributed systems based on isomorphisms of system computations. We have shown that this formalism is a generalization of the previously introduced and formalized notion of knowledge, which has proven to be useful in reasoning about distributed systems. Knowledge alone, however, is not adequate in describing formal semantics of loosely-coupled distributed applications, where the required loss of volition is too strict of a requirement. We show that the more general notion of belief is more appropriate to consider for describing the formal semantics of some of those applications where knowledge turns out to be too restrictive.

There are possible extensions of this work. For example, it would be of great interest to investigate a chain of belief of the form

$$P_1 \ \mathbf{bel}_{\alpha_1} \ (P_2 \ \mathbf{bel}_{\alpha_2} \ \ldots (P_n \ \mathbf{bel}_{\alpha_n} \ b)),$$

its relationship with process chains and how such a belief chain could be gained or lost. This was done for the case of knowledge in [3], however, it is more difficult in the case of belief.

Another related question is whether it is possible to define "common belief" in a distributed system. If it is, then apply it to some of the classical distributed protocols (such as coordinated attack problem) to obtain alternate or approximate solutions.

APPENDIX A

MATHEMATICAL PRELIMINARIES

We state some of the mathematical facts used in the thesis in this section. Proofs and more detailed treatment can be found in [22].

A.1 The Supremum Principle

Definition A.1. *Let E be a non-empty set of real numbers. A number $b \in \mathbb{R}$ is called an upper (resp. lower) bound for E if $x \leq b$ (resp. $x \geq b$) for all $x \in E$. In case such a b exists, we say that E is bounded from above (resp. below). If E has both an upper bound and a lower bound, then we say that E is bounded. By a supremum (or least upper bound) of E we mean a number $s \in \mathbb{R}$ such that*

i) s is an upper bound for E and

ii) if b is an upper bound for E, then $s \leq b$.

It is clear that E can have at most one supremum. If E has a supremum s, we write $s = supE$.

The infimum (or greatest lower bound) of E is defined similarly. It is a lower bound for E which is larger than every other lower bound for E.

Example A.1. *i) Let $E = \{x \in \mathbb{R} : x > 1\}$ Then any number ≤ 1 is a lower bound for E and $1 = infE$, but E has no upper bound. Note that $infE \notin E$.*

ii) Let $E = \{\frac{1}{n} : n \in \mathbb{Z}, n > 0\}$. Then $supE = 1 \in E$ while $infE = 0 \notin E$.

The Supremum Principle: Every non-empty subset of real numbers which is bounded from above has a supremum in $\mathbb{R}$ and every non-empty set of real numbers which is bounded from below has an infimum in $\mathbb{R}$.

The following proposition gives a slightly different characterization of the supremum.

Proposition A.1. *Let E be a non-empty subset of $\mathbb{R}$. Then $s = supE$ if and only if*

i) $x \leq s$ for all $x \in E$ and

ii) for every $\epsilon > 0, \epsilon \in \mathbb{R}$, there exists $x \in E$ such that $x > s - \epsilon$.

A.2 Sequences in $\mathbb{R}$

Definition A.2. *Let Y be a set and $f : \mathbb{N} \to Y$ be a function. Then f is called a sequence in Y. If $f(n) = x_n$, then x_n is called the nth term of the sequence. It is customary to write such a sequence as $(x_n)_{n=1}^{\infty}$ or as $(x_1, x_2, x_3, \dots)$ or as $x_1, x_2, x_3, \dots$. Also, we write $(x_n)_{n=1}^{\infty} \subseteq Y$ to indicate that the terms of the sequence are in the set Y.*

We will have occasion to consider functions whose domains of definition are of the form $\{n \in \mathbb{Z} : n \geq p\}$ for some fixed integer p. Such functions are also called sequences and are denoted as $(x_n)_{n=p}^{\infty}$. In fact, any function whose domain is a countable set is called a sequence.

Definition A.3. *A sequence $(x_n)_{n=1}^{\infty} \subseteq \mathbb{R}$ is said to converge if there exists $x \in \mathbb{R}$ such that*

$$\text{for every } \epsilon > 0, \text{ there exists } N_\epsilon \in \mathbb{N} \text{ such that } |x - x_n| < \epsilon \text{ for all } n \geq N_\epsilon.$$

In this case we will call x a limit for (x_n) and write $\lim_{n\to\infty} x_n = x$, or $x_n \to x$ as $n \to \infty$.

The limit of a sequence is unique.

Definition A.4. *A sequence* $(x_n) \subseteq \mathbb{R}$ *is said to be*

nondecreasing if $x_n \leq x_{n+1}$

nonincreasing if $x_n \geq x_{n+1}$

strictly increasing if $x_n < x_{n+1}$

strictly decreasing if $x_n > x_{n+1}$

for all $n \in \mathbb{N}$*. All such sequences are said to be monotone or, in the latter two cases, strictly monotone.*

Theorem A.1. *Any monotone sequence in* $\mathbb{R}$ *has a limit in* $\mathbb{R} \cup \{\infty, -\infty\}$*. In particular, a monotone sequence of real numbers converges if and only if it is bounded.*

Definition A.5. *Let* $(x_n) \subseteq \mathbb{R}$*. For each* $k \in \mathbb{N}$*, define*

$y_k = inf\{x_n : n \in \mathbb{N}, n \geq k\}$ *and* $z_k = sup\{x_n : n \in \mathbb{N}, n \geq k\}$*.*

Then y_k *is an increasing (nondecreasing) sequence in* $\mathbb{R}$ *and* z_k *is a decreasing (nonincreasing) sequence in* $\mathbb{R}$*. Therefore they both have limits in* $\mathbb{R} \cup \{\infty, -\infty\}$*. Let* $y = \lim_{k\to\infty} y_k = sup\{y_k : k \in \mathbb{N}\}$ *and* $z = \lim_{k\to\infty} z_k = inf\{z_k : k \in \mathbb{N}\}$*.*

The numbers y, z *are called the limit inferior of* (x_n) *and the limit superior of* (x_n) *respectively. We write*

$y = \liminf_{n\to\infty} x_n$*, and* $z = \limsup_{n\to\infty} x_n$*. Note that* $y \leq z$*.*

Another characterization of these numbers is as follows.

Theorem A.2. *Let* $(x_n) \subseteq \mathbb{R}$ *and let* $a, b \in \mathbb{R} \cup \{\infty, -\infty\}$. *Then*

i) $a = \liminf\limits_{n\to\infty} x_n$ *if and only if whenever* $\alpha < a$ *we have* $\{n \in \mathbb{N} : x_n < \alpha\}$ *is finite and whenever* $a < \beta$ *we have* $\{n \in \mathbb{N} : x_n < \beta\}$ *is infinite; and*

ii) $b = \limsup\limits_{n\to\infty} x_n$ *if and only if whenever* $\alpha < b$ *we have* $\{n \in \mathbb{N} : x_n > \alpha\}$ *is infinite and whenever* $b < \beta$ *we have* $\{n \in \mathbb{N} : x_n > \beta\}$ *is finite.*

The chief value of the lim inf and lim sup lies in the following theorem.

Theorem A.3. *Let* $(x_n) \subseteq \mathbb{R}$. *Then* $\lim x_n$ *exists in* $\mathbb{R} \cup \{\infty, -\infty\}$ *if and only if* $\limsup x_n = \liminf x_n$. *In this case,* $\lim x_n = \liminf x_n = \limsup x_n$.

Below are more properties of lim inf and lim sup.

Proposition A.2. *Let* (x_n) *and* (y_n) *be sequences of real numbers and* $c \in \mathbb{R}$. *Then*

i) $\liminf x_n + \liminf y_n \leq \liminf(x_n + y_n) \leq \mathit{limsup}(x_n + y_n) \leq \limsup x_n + \limsup y_n$

ii) if $\lim x_n = x \in \mathbb{R}$ *then* $\liminf(x_n + y_n) = x + \liminf y_n$ *and* $\limsup(x_n + y_n) = x + \limsup y_n$

iii) $\liminf(c \cdot x_n) = c \cdot \lim x_n$ *and* $\limsup(c \cdot x_n) = c \cdot \limsup x_n$ *if* $c \geq 0$

iv) $\liminf(c \cdot x_n) = c \cdot \limsup x_n$ *and* $\limsup(c \cdot x_n) = c \cdot \liminf x_n$, *if* $c \leq 0$.

APPENDIX B

NOTATION

x, y, z	system computations
p, q	processes
P, Q	process sets
D	the set of all processes in the system
$\bar{P}$	the complement of P
$\|x\|$	the length of computation x
$\|A\|$	the cardinality of the set A

b	predicate on system computations
b **at** x	the value of predicate b at computation x
$[P]_x$	the equivalence class containing x
$[P_b]_x$	the subset of $[P]_x$ where a given predicate b is true

Propositional connectives (in decreasing order of precedence)	
$\sim$	negation
$\wedge$, $\vee$	conjunction, disjunction
$\Rightarrow$, $\Leftrightarrow$	implication, equivalence

First order quantifiers	
$\forall$, $\exists$	universal, existential

BIBLIOGRAPHY

[1] Aumann, R.J. Agreeing to disagree. *Annals of Statistics*, 4:6, 1976.

[2] Chandy, K. M. and Misra, J. Drinking philosophers problem. *TOPLAS*, October 1984.

[3] Chandy, K. M. and Misra, J. How processes learn. *Distributed Computing* 1: 40-52, 1986.

[4] Cresswell, M. J. *Logics and languages.* London, Methuen and Co., 1973.

[5] Edwards, W. K. *Core Jini.* P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 2001.

[6] Dwork, C. and Moses, Y. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2): 156-186, 1990.

[7] Fagin, R., Halpern, J. Y., Moses Y. and Vardi, M. Y. Knowledge-based programs. *Distributed Computing*, 10(4): 199-225, 1997.

[8] Friedman, N. and Halpern, J. Y. Modeling belief in dynamic systems part II: revision and update. *Journal of AI Research*, 10: 117-167, 1999.

[9] Gettier, E. Is justified true belief knowledge? *Analysis* 23: 121-123, 1963.

[10] Graber, A., Burckert H-J., and Laux A. Terminological reasoning with knowledge and belief, in *Knowledge and Belief in Philosophy and Artificial Intelligence*, A. Laux and H. Wansing, Eds, Akademik Verlag, Berlin, 1995.

[11] Gray, C. G. and Cheriton, D. R. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*: 202–210, 1989.

[12] Halpern, J. Y. Reasoning about knowledge: a survey. *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay, C. J. Hogger, and J. A. Robinson, Eds, Oxford University Press, 4: 1-34, 1995.

[13] Halpern, Y. J. and Fagin, R. A formal model of knowledge, action and communication in distributed systems. *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing*: 224-236, 1985.

[14] Halpern, J. Y. and Moses, Y. Knowledge and common knowledge in a distributed environment. *Journal of the ACM* 37:3, 549-587, 1990.

[15] Hintikka, J. *Knowledge and Belief*, Cornell University Press, 1962.

[16] Lamport, L. Shostak, R. and Pease M. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3): 382-401, July 1982.

[17] Lynch N. and Fischer M. A lower bound for the time to assure interactive consistency. *Information Proc Letters* 14(4): 183-186, June 1982.

[18] McKee, B., Ehnebuske, D., and Rogers, D. UDDI version 2.0 API specification. available at http://www.uddi.org/specification.html, June 2001.

[19] Moses, Y. and Shoham, Y. Belief as defeasible knowledge. *Artificial Intelligence*, 64(2): 299-322, 1993.

[20] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, February 2001. Revision 2.4.2.

[21] Parikh, R. and Krasucki, P. Communication, consensus, and knowledge. *Journal of Economic Theory*, 52(1): 178-189, 1990.

[22] Stromberg, K. R. *An Introduction to Classical Real Analysis*, Wadsworth Inc, Belmont, California, 1981.

[23] Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043-1100. *Java Remote Method Invocation Specification*, revision 1.7, Java 2 SDK, v1.3.0 edition, December 1999.