Local and Temporal Predicates in Distributed Systems

BERNADETTE CHARRON-BOST
Ecole Polytechnique
and
CAROLE DELPORTE-GALLET and HUGUES FAUCONNIER
Université Paris 7

The definitions of the predicates Possibly ϕ and Definitely ϕ, where ϕ is a global predicate of a distributed computation, lead to the definitions of two predicate transformers P and D. We show that P plays the same role with respect to *time* as the predicate transformers K_i in knowledge theory play with respect to *space*. Pursuing this analogy, we prove that local predicates are exactly the fixed points of the K_i's while the stable predicates are the fixed points of P. In terms of the predicate transformers P and D, we define a new class of predicates that we call *observer-independent* predicates and for which the detection of Possibly ϕ and Definitely ϕ is quite easy. Finally, we establish a temporal counterpart to the knowledge change theorem of Chandy and Misra which formally proves that the global view of a distributed system provided by its various observations does not differ too much from its truth behavior.

Categories and Subject Descriptors: C.2.4 [**Computer–Communication Network**]: Distributed Systems—*distributed applications*; D.2.4 [**Software Engineering**]: Program Verification—*assertion checkers*; D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids*

General Terms: Theory, Verification

Additional Key Words and Phrases: Distributed computation, knowledge predicate, local predicate, observation, predicate, predicate transformer, temporal predicate

1. INTRODUCTION

Detection of global properties is a fundamental problem in asynchronous distributed systems. This problem arises in various contexts such as design, testing, or debugging of distributed systems. Chandy and Lamport [1985] propose a method for detecting *stable* predicates, i.e., predicates that continue to hold once they hold. Unfortunately, their detection scheme does not work for unstable predicates.

The major problem for detecting an arbitrary global predicate ϕ is that a distributed computation can be observed in many different (correct) manners, and

Authors' addresses: B. Charron-Bost, Laboratoire d'Informatique LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, France; email: charron@lix.polytechnique.fr; C. Delporte-Gallet and H. Fauconnier, Université Paris 7, LITP, 2 Place Jussieu, 75251 Paris Cedex 05; email: {cd;hf}@litp.ibp.fr.

according to the observation to which one refers, one will claim that ϕ is satisfied or not. Recently, Cooper and Marzullo [1991] addressed this issue by distinguishing the predicates which "possibly" hold, namely the predicates which hold in some observation, from those which "definitely" hold, i.e., which hold in all observations. In this way, for a given predicate ϕ, they define two new predicates referred to as "Possibly ϕ" and "Definitely ϕ." The remainder of their paper is devoted to the description of an algorithm for detecting Possibly ϕ and Definitely ϕ: all the possible global states which may be observed are recorded by making use of the vector clocks designed by Fidge [1988] and Mattern [1988].

In this article, our purpose is to investigate Possibly and Definitely as *predicate transformers*. Our viewpoint is similar to that of Chandy and Misra [1986]. Given a predicate ϕ, they define the predicate "process p_i knows ϕ" and deduce from this definition a predicate transformer referred to as K_i classically. A common feature of these various predicate transformers is that they "regularize" predicates: they transform a predicate into another one whose truth values are more uniform. For the K_i's, the improved uniformity is *spatial*; indeed, $K_i(\phi)$ takes the same truth value on all global states which are indistinguishable from the viewpoint of *process* p_i. For Possibly and Definitely, the uniformization is *temporal* since it occurs with respect to the *causal* structure, i.e., the appropriate analog of time in asynchronous distributed systems.

Investigating the properties satisfied by these predicate transformers reveals that Possibly is precisely the "temporal" counterpart to the K_i's. In this analogy, it turns out that stable predicates are the appropriate temporal analogs of local predicates. (Incidentally, it may be observed that, until now, mainly these two classes of predicates on distributed computations have been considered in the literature.) The idea that a distributed system comprising n processes may be canonically represented in an $n+1$-dimensional space, with one dimension per process and one dimension for time, has been introduced in Mattern [1988]. Our approach corroborates this picture since it places spatial and temporal structures on the same footing.

Advantageously, this viewpoint allows us to combine spatial and temporal transformers. In this way, we prove formally the basic axiom of *monotonicity of knowledge* which asserts that a process never forgets, i.e., once a process knows ϕ, it will forever know that ϕ held at some moment in the past.

Chandy and Misra [1986] discuss how knowledge is lost or gained. One of their key results shows that certain forms of knowledge cannot be lost by receiving nor gained by sending. Analogously, we study how an event can make a fact possible or definite and, conversely, how it can make uncertain a fact which was definite so far. Our result represents a temporal counterpart to the result of Chandy and Misra [1986] mentioned above. It guarantees that the use of *weak vector clocks* in Cooper and Marzullo's algorithm is valid.

The article is organized as follows. Section 2 contains our system model and presents the basic notions which will be used further. In Section 3, we briefly recall the definition of knowledge by Chandy and Misra [1986] and some basic results of their theory. In Section 4, the predicate transformers Possibly and Definitely are formally defined and are studied. In Section 5, we present our "temporal" counterpart to the theorem on knowledge change of Chandy and Misra. Section 6 contains concluding remarks.

2. SYSTEM MODEL

In this section, we describe a classical model of asynchronous distributed systems. Using this model, we formalize the notion of distributed computation and global state, and we state precisely how a distributed computation can be observed.

2.1 Distributed Computations

A distributed system consists of a finite set of processes $\{p_1, p_2, \cdots, p_n\}$ communicating only by messages. Each process runs a sequential program, whose execution is modeled by a finite sequence of *events*. Such a sequence is called a *local computation*. A process will be characterized by the set of its local computations. Clearly, this set is prefix closed.

Each event is either a *send* event, a *receive* event, or an *internal* event. A send event causes a message to be sent, and a receive event causes (1) a message to be received and (2) the local state to be updated by the content of the message. A send event and a receive event are said to *correspond* if the same message, sent in the send event, is received in the receive event. Internal events only affect the local process state. We assume that all events and all messages are different. Multiple occurrences of the same event or of the same message could be distinguished by affixing subscripts to them. We assume that communications are point-to-point, but we do not make any assumptions about the order in which messages are delivered. In particular, we do not assume that channels are FIFO.

As shown in Chandy [1988], any behavior of a distributed system is entirely described by the behavior of its processes. In other words, a computation c of a distributed system $\{p_1, \cdots, p_n\}$ is defined by the local computations $c[1], \cdots, c[n]$ performed by $p_1, \cdots, p_n$, respectively.

Since we model point-to-point communications, each communication event has at most one (unique) corresponding communication event. Moreover, every message which is received in a distributed computation must be sent in this computation (but not necessary vice-versa if we also consider computations where messages are still in transit).

Our model of distributed systems does not make use of time. However, events are ordered in the sense that certain events are known to cause other events. In distributed systems, causality results from sequential execution of single processes and from message passing between processes. It is formalized by the following definition stated originally by Lamport [1978]:

Definition 2.1.1. The *causality relation* $\prec$ is the smallest transitive relation on events that satisfies the following two conditions:

(1) if a and b are different events in the same local computation and if a occurs first then $a \prec b$;

(2) if s is the sending of a message and if r is the corresponding receive event then $s \prec r$.

Definition 2.1.2. A *computation* of a distributed system $\{p_1, \cdots, p_n\}$ is an n-tuple of local computations executed by $p_1, \cdots, p_n$, respectively, for which the causality relation $\prec$ is a strict partial order.

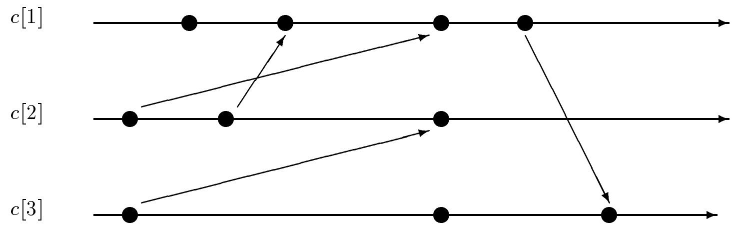

Fig. 1. Space-time diagram.

Note that, in general, $\prec$ is not a total order. Two distinct events are said to be *concurrent* if they are not ordered by $\prec$. An *antichain* of events is a set of events which are pairwise concurrent.

A convenient tool to visualize distributed computations is a *space-time diagram*. Figure 1 shows an example of a distributed computation comprising three processes. Lines represent the local computations of processes. Events are symbolized by dots. Messages are depicted as arrows connecting send events with their corresponding receive events.

2.2 Cuts and Consistent Cuts

At each step of a local computation, a process has a local state which results from its initial state and from the sequence of events that have occurred up to this step. Additionally, we assume that the local state of a process includes a description of all past events in the local computation that leads to this state. Hence, we model the state of a process by the local computation that it has executed for attaining this state. A global state in a distributed computation is defined as a collection of local states, one per process. Therefore, the global states in a distributed computation c are modeled by its *cuts*, which are defined as follows:

Definition 2.2.1. A *cut* of a distributed computation c is an n-tuple

$$\sigma = (\sigma[1], \cdots, \sigma[n])$$

where $\sigma[i]$ is a prefix of $c[i]$.

For any given computation c, a cut σ is unambiguously determined by the set $|\sigma|$ of events which occur in the prefixes defining the cut. Therefore, we may identify a cut σ in a given computation with the set of events $|\sigma|$, and for instance, we may say that some event belongs to a given cut.

The set of all cuts of a distributed computation is partially ordered by the relation $\sqsubseteq$ as defined below.

Definition 2.2.2. Two cuts σ and σ' of a computation satisfy $\sigma \sqsubseteq \sigma'$ if one of the following two equivalent conditions is satisfied:

(1) for any index i, $\sigma[i]$ is a prefix of $\sigma'[i]$;
(2) $|\sigma| \subseteq |\sigma'|$.

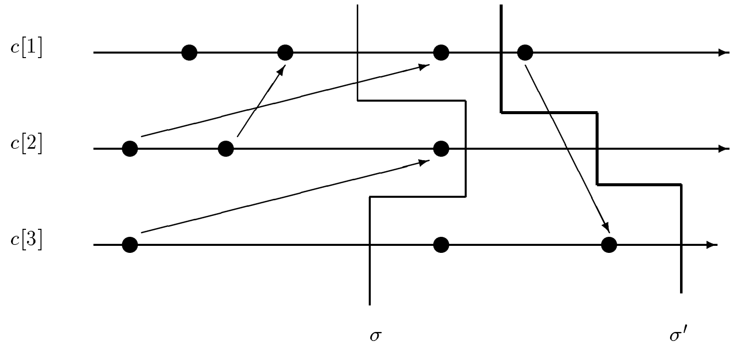

Fig. 2. Consistent versus inconsistent cuts.

Only certain global states can arise during the execution of a distributed computation. Equivalently, only certain cuts of a computation correspond to a possible (partial) behavior of the distributed system. These cuts are those which are consistent with regard to causality: if an event a occurs in such a cut then it must also contain all the causes of a. This leads to the following formal definition:

Definition 2.2.3. A cut σ of a distributed computation c is *consistent* if $|\sigma|$ is left-closed with respect to the causality relation, i.e.,

$$\forall a, b \ : \ a \in |c| \text{ and } b \in |\sigma| \ : \ (\text{if } a \prec b \text{ then } a \in |\sigma|).$$

Hence, a consistent cut of a distributed computation models a possible global state that the system may attain during this computation.

Observe that a consistent cut of a distributed computation is itself a computation. In Figure 2, σ is a consistent cut whereas σ' is not, since the message from p_1 to p_3 is received but is not sent in σ'.

The set of the consistent cuts of a computation c will be denoted by Γ_c. It immediately follows from the definition that the subsets of $|c|$ of the form $|\sigma|$ where $\sigma \in \Gamma_c$ are exactly the subsets of $|c|$ which are left-closed with respect to the causality relation $\prec$.

For any computation c, the set Γ_c is partially ordered by the relation $\sqsubseteq$ as defined above. The partially ordered set (or *poset*, for short) $(\Gamma_c, \sqsubseteq)$ has a greatest element, namely c, and a smallest element which is the empty cut denoted by $\perp$. For any two consistent cuts σ and σ' in Γ_c, $|\sigma| \cup |\sigma'|$ and $|\sigma| \cap |\sigma'|$ are clearly left-closed with respect to $\prec$. Therefore, there exist two unique consistent cuts of c denoted by $\sigma \cup \sigma'$ and $\sigma \cap \sigma'$ such that

$$|\sigma \cup \sigma'| = |\sigma| \cup |\sigma'| \text{ and } |\sigma \cap \sigma'| = |\sigma| \cap |\sigma'|.$$

They are respectively the least upper bound and the greatest lower bound of σ and σ' in the poset $(\Gamma_c, \sqsubseteq)$. This proves the following theorem that has been stated originally in Nielsen et al. [1981] (see also Mattern [1988]).

THEOREM 2.2.4. *The poset* $(\Gamma_c, \sqsubseteq)$ *is a complete lattice.*

The *down-sets* of Γ_c are defined as the subsets of Γ_c which are left-closed with respect to $\sqsubseteq$. When a down-set Γ of Γ_c admits a greatest element c', then clearly

$$\Gamma = \{\sigma \in \Gamma_c \ : \ \sigma \sqsubseteq c'\}$$

and thus $\Gamma = \Gamma_{c'}$.

In order to simplify notations, we will denote a cut and the set of events occurring in this cut by the same letter (i.e., we will write σ for $|\sigma|$). According to the remark which follows Definition 2.2.1, no confusion can arise from this identification.

2.3 Observations of a Distributed Computation

When observing a distributed computation, one observes how the global state of the computation is evolving. Thus any observation of a distributed computation will be modeled by a sequence of cuts. We rule out the observations which are not consistent with causality, i.e., those in which some event is observed before its causes. In this way, a local computation of a process is observed in a strictly sequential manner that preserves the original order. On the other hand, our observers are able to observe two concurrent events "simultaneously." More formally, we define an *observation* of a distributed computation as follows.

Definition 2.3.1. An *observation* of a distributed computation c is an increasing sequence $(\sigma_0, \sigma_1, \cdots, \sigma_p)$ of $(\Gamma_c, \sqsubseteq)$ which begins at $\sigma_0 = \perp$, ends at $\sigma_p = c$, and such that, for any index i, $\sigma_{i+1} \setminus \sigma_i$ is an antichain[1] of events.

In particular, a single event forms an antichain. If, for any index i, σ_{i+1} differs from σ_i by a single event e_{i+1} then $(\sigma_0, \sigma_1, \cdots, \sigma_p)$ is equivalent to the sequence of events $(e_1, e_2, \cdots, e_p)$. Since, for any k, $\sigma_k = \{e_1, \cdots, e_k\}$ is a consistent cut of c, the sequence $(e_1, \cdots, e_p)$ preserves the causality relation, i.e., if $e_i \prec e_j$ then $i < j$. In other words, $(e_1, \cdots, e_p)$ is a linear extension of the poset $(c, \prec)$. The observations of this kind correspond to "sequential" observers, in contrast to our observers who may perceive concurrent events in a single step.

By Szpilrajn's Theorem (cf., Szpilrajn [1930]), we know that the causality relation in a distributed computation is precisely the intersection of all its linear extensions.[2] Heuristically, this corresponds to what all sequential observations have in common. In that sense, sequential observations are sufficient for describing the underlying computation. This may explain why usually only sequential observations were considered in the literature until now (cf., Plotkin and Pratt [1990], Cooper and Marzullo [1991], or Schwarz and Mattern [1994], for instance). Nevertheless, certain characteristic features of a distributed computation cannot be captured by sequential observations as will be shown by an example in Section 4.1. For this reason, we have chosen to allow simultaneous observations of concurrent events.

In the following proposition, we state more precisely the connection between sequential and nonsequential observations.

[1]Recall that an *antichain* of a poset $(X, \prec)$ is any subset Y of X such that $\forall (a, b) \in Y^2 \ : \ \neg(a \prec b)$.

[2]A *linear extension* of a poset $(X, \prec)$ is any total order $<$ on X such that $\forall (a, b) \in X^2 \ : \ a \prec b \Rightarrow a < b$. Szpilrajn's Theorem asserts that, for any $(a, b) \in X^2$, $a \prec b$ iff $a < b$ for all linear extensions $<$ of $(X, \prec)$.

PROPOSITION 2.3.2. *Let σ and σ' be two cuts of a distributed computation such that $\sigma \sqsubseteq \sigma'$. Assume that σ is consistent. The cut σ' is consistent and $\sigma' \setminus \sigma$ is an antichain of events iff, for any event a of $\sigma' \setminus \sigma$, $\sigma \cup \{a\}$ is consistent.*

PROOF. First, assume that σ' is consistent and that $\sigma' \setminus \sigma$ is an antichain. Let a be an arbitrary event of $\sigma' \setminus \sigma$, and let p_i be the process on which a occurs. As $\sigma' \setminus \sigma$ is an antichain, necessarily

$$\sigma'[i] = \sigma[i]; a$$

(where ";" denotes the concatenation), and thus $\sigma \cup \{a\}$ is a cut. Since σ is consistent, in order to prove that $\sigma \cup \{a\}$ is consistent, it suffices to prove that any cause of a occurs in σ. Let x be an event such that $x \prec a$. Event a occurs in $\sigma' \setminus \sigma$ and thus occurs in σ'. By consistency of σ', x occurs in σ'. Since $\sigma' \setminus \sigma$ is an antichain, x occurs in fact in σ.

Conversely, assume that, for any event a of $\sigma' \setminus \sigma$, the cut $\sigma \cup \{a\}$ is consistent. If two events a and b in $\sigma' \setminus \sigma$ satisfy $a \prec b$ then $\sigma \cup \{b\}$ is not consistent. This proves that $\sigma' \setminus \sigma$ is an antichain. The consistency of σ' follows easily from this property and from the consistency of σ. $\square$

In the following, we consider an arbitrary distributed system $\{p_1, \cdots, p_n\}$ whose set of its computations is denoted by $\mathcal{C}$. The set of the observations of a distributed computation $c \in \mathcal{C}$ will be denoted by $\mathcal{O}(c)$. Note that the elements of $\mathcal{O}(c)$ do not all have the same length, in contrast to linear extensions whose common length is equal to the cardinality of c.

2.4 Predicates

We recall briefly some basic notations and definitions in logic. A *predicate* ϕ is a boolean function defined on $\mathcal{C}$ and hence can be seen as a property of possible global states. The assertion "ϕ holds at c" is denoted by $c \models \phi$.

The set of predicates will be denoted by $\mathcal{P}$. We define the binary relation $\Rightarrow$ on $\mathcal{P}$ as:

$$\phi \Rightarrow \psi \equiv \forall c \ : \ c \in \mathcal{C} \ : \ (\text{if } c \models \phi \text{ then } c \models \psi).$$

The set $\mathcal{P}$ is partially ordered by the relation $\Rightarrow$. With respect to $\Rightarrow$, any pair of predicates ϕ, ψ has a least upper bound $\phi \wedge \psi$ and a greatest lower bound $\phi \vee \psi$ which are defined by:

$$c \models \phi \wedge \psi \equiv c \models \phi \text{ and } c \models \psi$$

$$c \models \phi \vee \psi \equiv c \models \phi \text{ or } c \models \psi.$$

Moreover, the predicate which is always true (resp., false) is the least (resp., greatest) element of $(\mathcal{P}, \Rightarrow)$. This proves the following proposition.

PROPOSITION 2.4.1. *The poset $(\mathcal{P}, \Rightarrow)$ is a complete lattice.*

A predicate is *stable* if it continues to hold once it has been satisfied. This is formally expressed by the following condition:

$$\forall c, c' \ : \ (c, c') \in \mathcal{C}^2 \text{ and } c \sqsubseteq c' \ : \ (\text{if } c \models \phi \text{ then } c' \models \phi).$$

A *predicate transformer* is a function from $\mathcal{P}$ to $\mathcal{P}$. As in Dijkstra and Scholten [1990], a predicate transformer T is said to be *monotonic* if, for any two predicates ϕ and ψ in $\mathcal{P}$ such that $\phi \Rightarrow \psi$, it satisfies $T(\phi) \Rightarrow T(\psi)$. A predicate transformer T is said to be *finitely conjunctive* (resp., *finitely disjunctive*) if, for any two predicates ϕ and ψ in $\mathcal{P}$, it satisfies $T(\phi \wedge \psi) = T(\phi) \wedge T(\psi)$ (resp., $T(\phi \vee \psi) = T(\phi) \vee T(\psi)$). If T and T' are two predicate transformers on $\mathcal{P}$ then $T'T$ will denote the predicate transformer defined on $\mathcal{P}$ by:

$$\forall \phi \ : \ \phi \in \mathcal{P} \ : \ T'T(\phi) = T'(T(\phi)).$$

3. LOCAL AND KNOWLEDGE PREDICATES

In this section, we recall the definitions of local predicates and knowledge predicates given by Chandy and Misra [1986]. Because our purpose is to develop a "temporal" counterpart to these definitions, we do not deal with advanced aspects of knowledge theory such as common knowledge, for example. On the other hand, we point out the fact that locality and knowledge are equivalent notions.

3.1 Local Predicates

Let $P = \{p_1, \cdots, p_n\}$ denote the set of all the processes in a distributed system. Chandy and Misra [1986] define the predicates *local to a process set* Q ($Q \subseteq P$). Their definition captures the intuitive notion of predicates whose value is entirely controlled by the actions of the processes in Q. Before defining local predicates formally, they formalize the notion of computations which are "equivalent from the viewpoint of some process set."

Definition 3.1.1. For any two computations c and c' of a distributed system $c \sim_Q c'$ if

$$\forall i \ : \ p_i \in Q \ : \ c[i] = c'[i].$$

In other words, $c \sim_Q c'$ means that no process in Q is able to distinguish c from c'. Observe that $\sim_Q$ is an equivalence relation and that the equivalence class of c (with respect to this relation) is given by the local states of the processes in Q.

Note that $\sim_P$ is the equality relation. If $Q = \{p_i\}$ then the relation $\sim_Q$ is denoted by $\sim_i$ for short.

Definition 3.1.2. A predicate ϕ is *local* to a process set Q if its truth value depends only on the local states of the processes in Q, i.e.,

$$\forall c, c' \ : \ (c, c') \in \mathcal{C}^2 \text{ and } c \sim_Q c' \ : \ (c \models \phi \text{ iff } c' \models \phi).$$

Observe that a predicate local to $\{p_i, p_j\}$ is generally not local to both $\{p_i\}$ and $\{p_j\}$.[3]

Clearly a constant predicate is local to any process set. More interesting is the case of predicates which deal only with variables which are local to some process p_i; such predicates are local to p_i.

As an immediate consequence of Definition 3.1.2, we derive the following properties:

[3]In Chandy and Misra [1986], it is shown that any predicate local to disjoint process sets is constant.

PROPOSITION 3.1.3. *Given a process set Q in a distributed system, the set of predicates local to Q is closed under $\neg$, $\vee$, and $\wedge$.*

3.2 Knowledge Predicates

According to the common understanding of knowledge, for any predicate ϕ, some process p_i knows ϕ if p_i can conclude solely from its local state that ϕ holds. The definition of knowledge proposed by Chandy and Misra [1986] and Halpern and Moses [1990] relies on this basic idea. As a state local to a process set Q corresponds to an equivalence class with respect to $\sim_Q$, this definition of knowledge is based on the relation $\sim_Q$.

Let c be a computation of a distributed system, and let ϕ be a predicate.

Definition 3.2.1. *A process set Q* knows ϕ at c *if*

$$\forall c' \ : \ c' \in \mathcal{C} \text{ and } c' \sim_Q c \ : \ c' \models \phi.$$

In this way, from a given predicate ϕ, Chandy and Misra define a new predicate "Q knows ϕ" which will be denoted by $K_Q(\phi)$. This induces a *predicate transformer* K_Q. In the sequel, K_i is a shorthand for $K_{\{p_i\}}$. A predicate ψ is said to be a *knowledge* predicate for the process set Q if there exists some predicate ϕ such that $\psi = K_Q(\phi)$.

We next recall some basic properties of these predicate transformers which are straightforward consequences of Definition 3.2.1. These results have been given already by Chandy and Misra [1986] and by Sanders [1991].

Let Q be an arbitrary set of processes. With regard to the lattice structure $(\mathcal{P}, \Rightarrow)$, the predicate transformer K_Q obeys the following rules:

PROPOSITION 3.2.2. *K_Q is monotonic, i.e., if $\phi \Rightarrow \psi$ then $K_Q(\phi) \Rightarrow K_Q(\psi)$.*

Moreover, concerning the least upper bound and the greatest lower bound of two arbitrary predicates ϕ and ψ, we have the following properties:

PROPOSITION 3.2.3.

$$K_Q(\phi \wedge \psi) = K_Q(\phi) \wedge K_Q(\psi),$$
$$K_Q(\phi) \vee K_Q(\psi) \Rightarrow K_Q(\phi \vee \psi).$$

The following proposition states that, reasonably, only true facts are known:

PROPOSITION 3.2.4. *For any predicate ϕ, we have*

$$K_Q(\phi) \Rightarrow \phi.$$

Moreover, multiple levels of knowledge as "Q knows that Q knows ϕ" are redundant. This is formally expressed by the following proposition:

PROPOSITION 3.2.5. *The predicate transformer K_Q is idempotent:*

$$K_Q K_Q = K_Q.$$

It follows that the knowledge predicates for Q (which are the predicates in the image of K_Q, by definition) are in fact the fixed points of K_Q.

Combining Propositions 3.2.2, 3.2.4, and 3.2.5, we get an interesting characterization of the predicate $K_Q(\phi)$.

PROPOSITION 3.2.6. *The predicate $K_Q(\phi)$ is the weakest knowledge predicate for Q stronger than ϕ in the lattice $(\mathcal{P}, \Rightarrow)$.*

PROOF. Let ψ be a knowledge predicate for Q stronger than ϕ, i.e., such that $\psi \Rightarrow \phi$. As K_Q is monotonic, by applying K_Q, we get that $K_Q(\psi) \Rightarrow K_Q(\phi)$. Since K_Q is idempotent, and ψ is a knowledge predicate for Q, this implies

$$\psi \Rightarrow K_Q(\phi).$$

This shows that $K_Q(\phi)$ is weaker than any knowledge predicate for Q stronger than ϕ. On the other hand, by Proposition 3.2.4, the knowledge predicate $K_Q(\phi)$ is itself stronger than ϕ. $\square$

3.3 Locality and Knowledge

We conclude this section by proving that a predicate is local iff it is a knowledge predicate. This result reveals that local predicates are the fixed points of some predicate transformers (namely the K_Q's). As a matter of fact, the equivalence between locality and knowledge can be deduced from properties stated in Chandy and Misra [1986].

THEOREM 3.3.1. *A predicate ϕ is a knowledge predicate for a process set Q iff ϕ is local to Q.*

PROOF. First, we prove that if ϕ is a knowledge predicate for Q then ϕ is local to Q. Let c be a computation and assume that ϕ holds at c. Since ϕ is a fixed point of K_Q, $K_Q(\phi)$ holds at c, and thus we have:

$$\forall c' \ : \ c' \in \mathcal{C} \text{ and } c' \sim_Q c \ : \ c' \models \phi.$$

Since c and c' play symmetrical roles in Definition 3.1.2, this suffices to prove that ϕ is local to Q.

Then we show that if ϕ is local to Q then ϕ is a knowledge predicate for Q. Let c be a computation and assume that ϕ holds at c. Since ϕ is local to Q, we have

$$\forall c' \ : \ c' \in \mathcal{C} \text{ and } c' \sim_Q c \ : \ c' \models \phi.$$

It follows that $K_Q(\phi)$ holds at c. This proves that ϕ implies $K_Q(\phi)$. By Proposition 3.2.4 it follows that $K_Q(\phi) = \phi$, i.e., ϕ is a fixed point of K_Q. $\square$

Consequently, knowledge predicates inherit properties satisfied by local predicates and vice-versa. As an example, $\neg K_Q(\phi)$ (which is not equivalent to $K_Q(\neg\phi)$) is local to Q. Thus $\neg K_Q(\phi)$ is a fixed point of K_Q, i.e.,

$$\neg K_Q(\phi) = K_Q\left(\neg K_Q(\phi)\right).$$

4. TEMPORAL PREDICATES

In this section, we will consider two other predicate transformers in order to express temporal properties of a distributed system. We will characterize the stable properties in terms of one of these transformers. Furthermore, we will define a new class of predicates (more general than the class of stable predicates) whose detection is easy.

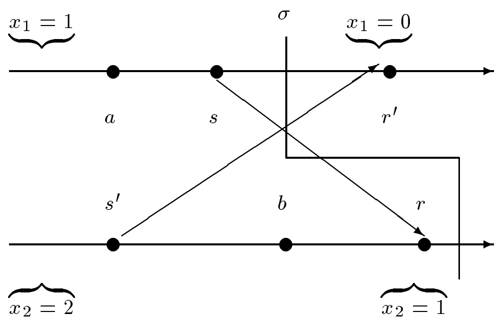

Fig. 3. A distributed computation.

4.1 Possibly and Definitely

Detecting global predicates is a fundamental issue in designing, testing, or debugging distributed systems. The problem is to observe the different global states reached by the system and to test whether these global states satisfy certain properties. However, an observation only reveals a subset of the possible global states. Consequently, two observers of the same distributed computation may observe different sets of global states. The opinions of these observers concerning the satisfaction of a given predicate may diverge.

For example, consider the distributed computation c shown in Figure 3 and the (sequential) observations O and O' defined by:

$$O = (s', b, a, s, r, r') \text{ and } O' = (a, s, s', r', b, r).$$

The initial values of the local variables x_1 and x_2 are 1 and 2, respectively. When r' (resp., r) occurs, x_1 (resp., x_2) is assigned to 0 (resp., 1). Only one consistent cut σ of c satisfies "$x_1 = x_2$." This consistent cut σ occurs in O (after r) but not in O'. The observer that makes observation O will detect that $x_1 = x_2$ at some moment whereas the observer that makes observation O' will conclude that the local variables x_1 and x_2 are never equal.

Cooper and Marzullo [1991] address this issue by introducing two new predicate transformers P (for Possibly) and D (for Definitely) defined as follow:

Definition 4.1.1.

$$c \models P(\phi) \equiv \exists \sigma_0, \cdots, \sigma_p \ : \ (\sigma_0, \cdots, \sigma_p) \in \mathcal{O}(c) \ : \ (\exists i \ : \ i \in \{0, \cdots, p\} \ : \ \sigma_i \models \phi).$$

$$c \models D(\phi) \equiv \forall \sigma_0, \cdots, \sigma_p \ : \ (\sigma_0, \cdots, \sigma_p) \in \mathcal{O}(c) \ : \ (\exists i \ : \ i \in \{0, \cdots, p\} \ : \ \sigma_i \models \phi).$$

In other words, $P(\phi)$ states that there is at least one observation in which ϕ held at least once, and $D(\phi)$ states that during any observation ϕ held at least once.

Since the set of observations $\mathcal{O}(c)$ does not depend on the maximal computation in which c is embedded, i.e., does not depend on the future of c, predicates $P(\phi)$ and $D(\phi)$ are indeed predicates on the global states of a distributed system.[4]

[4]In contrast to the predicates defined in Panangaden and Taylor [1992] whose truth value at a computation c depends on the run in which c is embedded.

Observe that the definition of $P(\phi)$ can be stated equivalently as

$$c \models P(\phi) \equiv \exists c' \ : \ c' \in \Gamma_c \ : \ c' \models \phi$$

without referring to the set $\mathcal{O}(c)$. This shows that the predicate $P(\phi)$ does not depend on the way we model an observation. This is not the case of predicate $D(\phi)$: the predicate $D(\phi)$ that results from Definition 4.1.1, where sequential and nonsequential observations are taken into account, is not identical to the one induced by executing the computation exclusively in a sequential fashion as in Cooper and Marzullo [1991] and Schwarz and Mattern [1994]. Consider, for example, a distributed system with two processes p_1 and p_2; each of them has a local variable (say x_1 for p_1 and x_2 for p_2). Processes p_1 and p_2 perform the assignments $x_1 := x_1 + 1$ and $x_2 := x_2 - 1$, respectively. Assume that initially $x_1 = 1$ and $x_2 = 2$ and consider the global predicate $\phi \equiv (x_1 = x_2)$. Denote the unique maximal computation by c. According to our definition of $\mathcal{O}(c)$, $D(\phi)$ does not hold whereas ϕ is declared to hold definitely in Cooper and Marzullo [1991] and Schwarz and Mattern [1994]. Indeed, if p_1 and p_2 act simultaneously, then we never observe that $x_1 = x_2$. Therefore it seems reasonable to extend the notion of observation as done in Definition 2.3.1.

When regarding P and D as predicate transformers, it is interesting to study the behavior of P and D with respect to the lattice structure $(\mathcal{P}, \Rightarrow)$. As an immediate consequence of Definition 4.1.1, we get:

PROPOSITION 4.1.2. *The predicate transformers D and P are monotonic, i.e., if $\phi \Rightarrow \psi$ then $D(\phi) \Rightarrow D(\psi)$ and $P(\phi) \Rightarrow P(\psi)$.*

This implies the following compatibilities of P and D with the operators $\wedge$ and $\vee$.

PROPOSITION 4.1.3. *The predicate transformer P is finitely disjunctive, i.e., for any two predicates ϕ and ψ,*

$$P(\phi \vee \psi) = P(\phi) \vee P(\psi).$$

Moreover, the following implications hold:

$$\begin{aligned} P(\phi \wedge \psi) &\Rightarrow P(\phi) \wedge P(\psi), \\ D(\phi) \vee D(\psi) &\Rightarrow D(\phi \vee \psi), \\ D(\phi \wedge \psi) &\Rightarrow D(\phi) \wedge D(\psi). \end{aligned}$$

PROOF. We just give a proof of the first relation which, incidentally, makes the detection of $P(\phi)$ easier for a wide range of predicates ϕ (cf., Proposition 4.3.3).

Applying the fact that P is monotonic (Proposition 4.1.2) we get that $P(\phi)$ as well as $P(\psi)$ imply $P(\phi \vee \psi)$. Therefore we have:

$$P(\phi) \vee P(\psi) \Rightarrow P(\phi \vee \psi).$$

Conversely, assume that $P(\phi \vee \psi)$ holds at a computation c, i.e., that there exists a consistent cut c' of c such that $\phi \vee \psi$ holds at c'. We deduce that $P(\phi)$ or $P(\psi)$ holds at c according to $c' \models \phi$ or $c' \models \psi$. This shows that

$$P(\phi \vee \psi) \Rightarrow P(\phi) \vee P(\psi),$$

and completes the proof of the first relation. □

Clearly $D(\phi)$ implies $P(\phi)$. Moreover, since any observation of c ends in c and so contains c, any predicate ϕ implies $D(\phi)$:

PROPOSITION 4.1.4. *For any predicate ϕ, we have*

$$\phi \Rightarrow D(\phi) \text{ and } D(\phi) \Rightarrow P(\phi).$$

This proposition may be used to prove:

PROPOSITION 4.1.5. *The predicate transformers D and P satisfy the following equalities:*

$$PP = P,\ DD = D,\ \text{and } DP = PD = P.$$

PROOF. We show that P is idempotent; the other properties can be proven similarly. Applying Proposition 4.1.4 to the predicate $P(\phi)$, we get that $P(\phi)$ implies $PP(\phi)$, i.e.,

$$P(\phi) \Rightarrow PP(\phi).$$

Conversely, assume that $PP(\phi)$ holds at c, i.e., that there exists a consistent cut c' of c such that $P(\phi)$ holds at c'. This means that ϕ holds at a consistent cut c'' of c'. Since c'' is also a consistent cut of c we deduce that $P(\phi)$ holds at c. In other words

$$PP(\phi) \Rightarrow P(\phi).$$

Consequently we have $PP = P$. □

4.2 When ϕ and $P(\phi)$ Coincide

Considering the hierarchy of predicates $\phi \Rightarrow D(\phi) \Rightarrow P(\phi)$, it is quite natural to study the predicates for which this hierarchy is not strict. We shall first consider the (stronger) equation

$$\phi = P(\phi). \tag{1}$$

Observe that the solutions of (1) are the fixed points of P. Since P is idempotent, a predicate is a fixed point of P iff it belongs to the image of P.

THEOREM 4.2.1. *A predicate ϕ satisfies the equation $\phi = P(\phi)$ iff ϕ is a stable predicate.*

PROOF. First, we prove that any stable predicate is a fixed point of P. Assume that $P(\phi)$ holds at c, i.e., there exists a consistent cut c' of c such that ϕ holds at c'. Since ϕ is stable it holds at c, too. It follows that $P(\phi)$ implies ϕ. By Proposition 4.1.4, ϕ implies $P(\phi)$, and so $P(\phi)$ and ϕ are equal.

Conversely, let ϕ be a fixed point of P. Let c and c' be two computations of a distributed system such that c' is a consistent cut of c. If ϕ holds at c' then $P(\phi)$ holds at c, and so ϕ holds at c. In other words, ϕ is a stable predicate. □

From Proposition 4.1.4 and Theorem 4.2.1, it follows that $P(\phi)$ and $D(\phi)$ are equal for any stable predicate ϕ. This corollary of Theorem 4.2.1 appears already in Katz and Peled [1990] and Schwarz and Mattern [1994].

Until recently, mainly stable properties of distributed systems (such as deadlock or termination) were investigated in the literature. Theorem 4.2.1 may explain why the predicate transformers P and D were not considered in this framework. However, for debugging purposes, it is required to detect global properties that are not stable. To this end, we could use a *snapshot* algorithm for recording a global state of the underlying system and then check whether some global property holds at this state. Unfortunately, in this way, it is not possible to distinguish between a global state that the system has *definitely* reached and one that it has *possibly* reached. A snapshot algorithm allows us to record a *possible* state, that is, a state which might not occur in a particular observation but which does occur in another possible observation. In terms of the predicate transformer P, for a given predicate ϕ, a snapshot algorithm may detect $P(\phi)$. Observe that it may be the case that ϕ did actually hold, but the snapshot algorithm does not detect it. When ϕ is stable, Theorem 4.2.1 shows that one detects ϕ safely in this way (a simpler proof of this fact is given in Chandy and Lamport [1985]). This motivates the practical importance of snapshot algorithms.

Together with the idempotency of P, Theorem 4.2.1 shows that $P(\phi)$ is stable, even if ϕ is not stable. Because P is monotonic, this implies (by a proof similar to the one of Proposition 3.2.6):

Proposition 4.2.2. *The predicate $P(\phi)$ is the strongest stable predicate weaker than ϕ in the lattice of predicates $(\mathcal{P}, \Rightarrow)$.*

From Proposition 4.2.2, it follows that the predicate transformer P is the *sin* (strongest invariant) predicate transformer defined by Lamport [1990].

At this point, a deep analogy arises between the predicate transformers P and K_i. First these predicate transformers satisfy common basic properties that are summarized in Figure 4.[5] Note that the analogy is even more striking when considering the dual of P,[6] namely $\tilde{P}$, instead of P. A fixed point of the predicate transformer $\tilde{P}$ is the negation of a stable predicate. Such a predicate is characterized by the fact that if it holds then it held in the whole past. This is the reason why we have called these predicates *inherited* predicates.

Besides these common properties, we can define locality to process p_i as an invariance property with respect to any move outside p_i, whereas stability expresses invariance with respect to moves in the future. According to the characterizations of $K_i(\phi)$ and $P(\phi)$ given in Propositions 3.2.6 and 4.2.2, the predicate transformer K_i can be viewed as a *spatial-regularizing* transformer whereas P appears as a *temporal-regularizing* transformer. With regard to this analogy, the equivalence classes $H_{c_i} = \{\sigma \in \mathcal{C} \ : \ \sigma[i] = c_i\}$ (where c_i is some local computation of process p_i) play the same role in the study of K_i as the down-sets of consistent cuts $\Gamma_c = \{\sigma \in \mathcal{C} \ : \ \sigma \sqsubseteq c\}$ play in the study of P. However, an important difference is that $K_i(\phi)$ has the same truth value on each equivalence class H_{c_i}, whereas it may be the case that $P(\phi)$ does not uniformly hold in a down-set Γ_c (we only have that if $P(\phi)$ does not hold at c then it never holds in Γ_c). This comes from the fact that H_{c_i} is defined in terms of an *equivalence* relation (namely $\sim_i$) whereas Γ_c is defined

[5] From the first three lines of this table, we deduce that K_i and $\tilde{P}$ are closure operators.
[6] The dual of a predicate transformer T, denoted generally by $\tilde{T}$, is defined by:

$$c \models \tilde{T}(\phi) \equiv c \models \neg (T(\neg\phi)) .$$

Consequently, $c \models \tilde{P}(\phi)$ iff ϕ holds at all the consistent cuts of c.

	K_i	P	$\tilde{P}$
Monotonicity	monotonic	monotonic	monotonic
Comparison with ϕ	$K_i(\phi) \Rightarrow \phi$	$\phi \Rightarrow P(\phi)$	$\tilde{P}(\phi) \Rightarrow \phi$
Idempotence	$K_i K_i = K_i$	$PP = P$	$\tilde{P}\tilde{P} = \tilde{P}$
Junctivity	finitely conjunctive	finitely disjunctive	finitely conjunctive
Fixed point	local to p_i	stable	inherited

Fig. 4. Common properties satisfied by P, $\tilde{P}$, and K_i.

in terms of the *partial order* $\sqsubseteq$.

4.3 When $P(\phi)$ and $D(\phi)$ Coincide

Detecting $D(\phi)$ seems to be computationally very expensive because it requires inspection of all observations. In the worst case, detecting $P(\phi)$ is as expensive as detecting $D(\phi)$. If a predicate ϕ satisfies $P(\phi) = D(\phi)$ then the fact that ϕ held in some observation guarantees that ϕ held in any observation. If Possibly and Definitely are equal then the detection of $D(\phi)$ and $P(\phi)$ is much easier. Consequently, the class of predicates for which Possibly and Definitely coincide is of great practical importance.

Definition 4.3.1. A predicate ϕ is said to be *observer-independent* if $P(\phi)$ and $D(\phi)$ are equal.

According to the observation following Theorem 4.2.1, a stable predicate is observer-independent. But do there exist observer-independent predicates which are not stable? It turns out that this is the case, and the following proposition provides an important example of such predicates.

PROPOSITION 4.3.2. *A predicate that is local to one process is observer-independent, i.e.,*

$$\forall i \ : \ i \in \{1, \cdots, n\} \ : \ DK_i = PK_i.$$

PROOF. Let ϕ be a predicate local to process p_i. Assume that $P(\phi)$ holds at a given computation c, i.e., that there exists a consistent cut c' of c such that ϕ holds at c'. Let $(\sigma_0, \cdots, \sigma_p)$ be an observation of c. The projection on p_i of any cut in this observation is a prefix of the local computation $c[i]$ and is a prefix of $c'[i]$, or $c'[i]$ is a prefix of it. Moreover, the sequence $(\sigma_j[i])_{j=0}^{p}$ is nondecreasing. If, for some index k, $\sigma_k[i]$ is a strict prefix of $c'[i]$, and $c'[i]$ is a strict prefix of $\sigma_{k+1}[i]$, then $\sigma_{k+1} \setminus \sigma_k$ contains at least two events processed by p_i. This contradicts that $\sigma_{k+1} \setminus \sigma_k$ is an antichain. Progress on p_i is therefore continuous in each observation, and there exists at least one index k such that $\sigma_k[i] = c'[i]$. With regard to $\sim_i$, σ_k and c' are equivalent. As ϕ is local to p_i, the fact that ϕ holds at c' implies that ϕ holds at σ_k. Consequently, ϕ holds in any observation of c, i.e., $D(\phi)$ holds at c.

This proves that $P(\phi)$ implies $D(\phi)$. Since $D(\phi)$ implies $P(\phi)$ (Proposition 4.1.4), this concludes the proof. $\square$

Thus the fixed points of the predicate transformers P and K_i (i.e., the stable and the local predicates) are observer-independent. This stresses the analogy between these predicate transformers.

By definition of a stable predicate, the disjunction and the conjunction of two stable predicates are stable. But what about observer-independent predicates? In general, the conjunction of two observer-independent predicates is not observer-independent. To see this, consider the computation c defined at the beginning of Section 4.1 (cf., Figure 3) and the predicate $\psi \equiv (x_1 = 1) \wedge (x_2 = 1)$. Clearly, $P(\psi)$ holds at c, but if p_1 changes its local state before p_2 then we never observe the satisfaction of ψ. It follows that $D(\psi)$ does not hold at c. Though ψ is the conjunction of two observer-independent (because local to one process) predicates, ψ is not observer-independent. As for the disjunction, we have the following result:

PROPOSITION 4.3.3. *The disjunction of any two observer-independent predicates is observer-independent.*

PROOF. Let ϕ and ψ be two observer-independent predicates. Then we have

$$P\,(\phi \vee \psi) = P(\phi) \vee P(\psi) = D(\phi) \vee D(\psi)$$

and

$$D(\phi) \vee D(\psi) \Rightarrow D(\phi \vee \psi).$$

Indeed, the first equality follows from the fact that P is finitely disjunctive (Proposition 4.1.3); the second holds because ϕ and ψ are observer-independent; and the implication also follows from Proposition 4.1.3. Since Definitely implies Possibly, we have $P(\phi \vee \psi) = D(\phi \vee \psi)$, i.e., $\phi \vee \psi$ is observer-independent. $\square$

As a consequence of Propositions 4.3.2 and 4.3.3, predicates of the form $K_{i_1}(\phi_{i_1}) \vee \cdots \vee K_{i_r}(\phi_{i_r})$ are observer-independent. Detecting whether such predicates hold definitely or possibly is quite easy since it just requires us to inspect the global states that an arbitrary observation comprises.

4.4 Locality and Stability

To understand the semantics of P, D, and K_i better, we now study the commutation properties of these predicate transformers. In other words, we compare the four predicates $DK_i(\phi)$, $PK_i(\phi)$, $K_iD(\phi)$, and $K_iP(\phi)$ arising from an arbitrary predicate ϕ. By Proposition 4.3.2, we know that the first two predicates, namely, $DK_i(\phi)$ and $PK_i(\phi)$, are equal. Moreover, by Propositions 3.2.2 and 4.1.4, $K_iD(\phi)$ implies $K_iP(\phi)$. As for $DK_i(\phi)$ and $K_iD(\phi)$, we have the following implication:

PROPOSITION 4.4.1. *For any predicate ϕ, we have*

$$DK_i(\phi) \Rightarrow K_iD(\phi).$$

PROOF. Let c be a computation. Assume that the predicate $DK_i(\phi)$ which coincides with $PK_i(\phi)$ holds at c, i.e., that there exists a consistent cut σ of c such that $\sigma \models K_i(\phi)$. Let c' be a distributed computation which is equivalent to c from p_i's viewpoint, i.e., $c' \sim_i c$, and let $(\sigma_0, \cdots, \sigma_p)$ be an observation of c'. The projection on p_i of this observation forms a nondecreasing sequence of prefixes of $c'[i] = c[i]$. Moreover two consecutive elements of this sequence differ in at most one event. Since σ is a consistent cut of c, $\sigma[i]$ is a prefix of $c[i]$, and there exists an index j such that $\sigma_j[i] = \sigma[i]$. Since p_i knows ϕ at σ it follows that $\sigma_j \models \phi$. Consequently, ϕ holds definitely at any computation which is equivalent to c with regard to p_i, i.e., $c \models K_i D(\phi)$. □

Proposition 4.4.1 shows that if possibly (or definitely) p_i knew ϕ then p_i knows that ϕ held definitely. Informally speaking, this means that if p_i knows ϕ at c then it may be the case that p_i does not know ϕ in the future of c, but p_i does not forget that ϕ has been true. Consequently, *processes do not forget, and so knowledge is never lost.* Interestingly, this property that we have formally proved in our model appears as a basic assumption in the timed-runs semantics of Halpern and Moses [1990].

From Proposition 4.4.1 and the observations preceding this proposition, it follows that, for any predicate ϕ, we have

$$DK_i(\phi) = PK_i(\phi) \text{ and } PK_i(\phi) \Rightarrow K_i D(\phi) \Rightarrow K_i P(\phi). \tag{2}$$

If ϕ is a stable predicate then $P(\phi) = \phi$ and, in this case, (2) gives

$$PK_i(\phi) \Rightarrow K_i(\phi).$$

Therefore $PK_i(\phi) = K_i(\phi)$, i.e., $K_i(\phi)$ is a stable predicate.

On the other hand, if ϕ is local to process p_i then $K_i(\phi) = \phi$, and (2) gives

$$D(\phi) = P(\phi) \text{ and } P(\phi) \Rightarrow K_i D(\phi) \Rightarrow K_i P(\phi).$$

By Proposition 3.2.4, we deduce that

$$D(\phi) = P(\phi) = K_i D(\phi) = K_i P(\phi),$$

i.e., $D(\phi)$ and $P(\phi)$ are local to process p_i. These results are summarized in the following corollary:

COROLLARY 4.4.2. *Let ϕ be a predicate.*

(1) If ϕ is stable then $K_i(\phi)$ is stable.
(2) If ϕ is local to process p_i then $D(\phi)$ and $P(\phi)$ are local to process p_i.

5. A TEMPORAL COUNTERPART TO THE THEOREM ON KNOWLEDGE CHANGE

Chandy and Misra [1986] have shown that certain forms of knowledge cannot be lost by sending nor gained by receiving. In order to bring the analogy between K_i and P into line as much as possible, we would like to establish a "temporal" counterpart to this key result. For that, we introduce the notion of an event that *potentially affects a predicate* defined originally in Katz and Peled [1990]. Based on this notion, we give an equivalent statement to the knowledge change theorem of Chandy and Misra [1986] that admits a natural counterpart for the predicate transformers P and D. In our presentation, all these results are proved by similar technical arguments. Finally, as an application of our result, we show that *weak vector clocks* are valid for detecting $P(\phi)$ and $D(\phi)$.

5.1 Events that Potentially Affect a Predicate

Let ϕ be a predicate defined on the computations of a distributed system. We will say that an event e *can occur after* a computation c if there exists a computation c' such that c is a consistent cut of c' and $c' \setminus c = \{e\}$. In this case, c' will be denoted by $c' = c \cup \{e\}$.

Definition 5.1.1. Let ϕ be a predicate defined on the computations of a distributed system.

(1) An event e is said to *affirm ϕ potentially* if there exists a computation c after which e can occur and such that

$$c \models \neg\phi \text{ and } c \cup \{e\} \models \phi.$$

(2) An event e is said to *reject ϕ potentially* if there exists a computation c after which e can occur and such that

$$c \models \phi \text{ and } c \cup \{e\} \models \neg\phi.$$

(3) An event e is said to *affect ϕ potentially* if it affirms or rejects ϕ potentially.

Let $\mathcal{A}_\phi^+$ (resp., $\mathcal{A}_\phi^-$, $\mathcal{A}_\phi$) denote the set of events that affirm (resp., reject, affect) ϕ. Observe that $\mathcal{A}_\phi = \mathcal{A}_\phi^+ \cup \mathcal{A}_\phi^-$. However, $(\mathcal{A}_\phi^+, \mathcal{A}_\phi^-)$ does not generally form a partition of $\mathcal{A}_\phi$ because an event may both affirm and reject a predicate.

Stability of a predicate ϕ can be expressed by $\mathcal{A}_\phi^- = \emptyset$. But what about $\mathcal{A}_\phi$ when ϕ is a local predicate? Interestingly, it is possible to characterize the predicates which are local to a process set by means of the sets $\mathcal{A}_\phi$.

Proposition 5.1.2. *A predicate ϕ defined on the computations of a distributed system is local to a process set Q iff $\mathcal{A}_\phi$ is included in the set of events which occur in the processes of Q.*

Proof. Assume that ϕ is local to a process set Q. Let e be an event on a process outside Q, and let c be an arbitrary computation such that e can occur after c. The computations c and $c \cup \{e\}$ are isomorphic with respect to Q. Since ϕ is local to Q, it has the same truth value at c and at $c \cup \{e\}$. Hence, e cannot affect ϕ, i.e., $e \notin \mathcal{A}_\phi$.

Conversely, assume that $\mathcal{A}_\phi$ is included in the set of events which occur in the processes of Q. Any equivalence class of computations with respect to Q admits a unique smallest element (to see this, consider the intersection of two smallest elements). Let σ be such a computation in the equivalence class

$$[\sigma]_Q = \{c \in \mathcal{C} \;:\; c \sim_Q \sigma\}.$$

For any computation c in $[\sigma]_Q$, we have $\sigma \sqsubseteq c$, and there exists a sequence of events $e_1, \cdots, e_p$ such that $\sigma, \sigma \cup \{e_1\}, \cdots, \sigma \cup \{e_1, \cdots, e_i\}, \cdots, \sigma \cup \{e_1, \cdots, e_p\} = c$ forms an increasing chain of computations from σ to c. Moreover, every event e_i is executed by a process outside Q, and so cannot affect ϕ. It follows that ϕ has the same truth value at σ and at c. Consequently, ϕ is constant on each equivalence class $[\sigma]_Q$, i.e., is local to Q. □

In particular, when ϕ is local to one process, $\mathcal{A}_\phi$ is totally ordered by the causality relation, and ϕ is observer-independent (cf., Proposition 4.3.2). This gives an alternative proof of a similar result stated in Katz and Peled [1990].

5.2 How Local Predicates Change

Now we are in position to express the theorem on knowledge change of Chandy and Misra [1986] in terms of the sets $\mathcal{A}_\phi^+$, $\mathcal{A}_\phi^-$, and $\mathcal{A}_\phi$. First we recall their key result. As before, the set of the processes of the distributed system that we consider is denoted by $P = \{p_1, \cdots, p_n\}$.

Theorem 5.2.1 [Chandy and Misra 1986, p. 48]. *Let ϕ be a predicate local to $P \setminus \{p_i\}$, and let e be an event which occurs in p_i. Let c be a computation after which e can occur.*
(1) e is a receive event: if $c \models K_i(\phi)$ then $c \cup \{e\} \models K_i(\phi)$, i.e., knowledge is not lost.
(2) e is a send event: if $c \cup \{e\} \models K_i(\phi)$ then $c \models K_i(\phi)$, i.e., knowledge is not gained.
(3) e is an internal event: $c \models K_i(\phi)$ iff $c \cup \{e\} \models K_i(\phi)$, i.e., knowledge is neither lost nor gained.

Observe that according to Proposition 5.1.2, the assumptions of Theorem 5.2.1 guarantee that $e \notin \mathcal{A}_\phi$. Using this fact, we are going to derive the following statement, which implies Theorem 5.2.1 and may be considered as a mere variant of it. This variant has the advantage of admitting a "temporal counterpart" as will be shown in Section 5.3.

Theorem 5.2.2. *Let ϕ be a predicate, and let e be an event.*
(1) e is a receive event: if $e \in \mathcal{A}_{K_i(\phi)}^-$ then $e \in \mathcal{A}_\phi^-$.
(2) e is a send event: if $e \in \mathcal{A}_{K_i(\phi)}^+$ then $e \in \mathcal{A}_\phi^+$.
(3) e is an internal event: if $e \in \mathcal{A}_{K_i(\phi)}$ then $e \in \mathcal{A}_\phi$.

We will use the following technical lemma, the proof of which we omit since it is trivial.

Lemma 5.2.3. *Let c be a computation, and let e be an event such that e can occur after c.*
(1) If e is a receive event or an internal event, for any computation σ which is equivalent to $c \cup \{e\}$ with respect to some process p_i, $\sigma \setminus \{e\}$ is a computation and $\sigma \setminus \{e\} \sim_i c$.
(2) If e is a send event or an internal event, for any computation σ which is equivalent to c with respect to some process p_i, $\sigma \cup \{e\}$ is a computation and $\sigma \cup \{e\} \sim_i c \cup \{e\}$.

Proof of Theorem 5.2.2. Let us prove assertion (1). Let e be a receive event in $\mathcal{A}_{K_i(\phi)}^-$. Thus there exists a computation c after which e can occur such that

$$c \models K_i(\phi) \text{ and } c \cup \{e\} \models \neg K_i(\phi).$$

Then there exists a computation σ such that $\sigma \sim_i c \cup \{e\}$ and $\sigma \models \neg\phi$. By Lemma 5.2.3, $\sigma \setminus \{e\}$ is a computation which is equivalent to c with respect to p_i. Then $\sigma \setminus \{e\} \models \phi$ and $\sigma \models \neg\phi$. This proves that e rejects ϕ potentially.

Proofs of assertions (2) and (3) are analogous and left to the reader. □

Observe that it may be the case that an event affects potentially the knowledge of a predicate ϕ without being able to affect ϕ directly. Figure 3 shows such a case where $\phi \equiv (x_2 = 2)$, in contrast to $K_1(\phi)$, cannot be affected by event s.

5.3 How Temporal Predicates Change

In the preceding section, we showed how the key result in Chandy and Misra [1986] states a relationship between the sets of events $\mathcal{A}_\phi$ and $\mathcal{A}_{K_i(\phi)}$. An analogous result holds for the predicate transformers P and D.

THEOREM 5.3.1. *For any predicate ϕ, we have*

$$\mathcal{A}_{P(\phi)} \subseteq \mathcal{A}_\phi \textit{ and } \mathcal{A}_{D(\phi)} \subseteq \mathcal{A}_\phi.$$

The proof of this theorem relies on the following technical lemma which is the counterpart of Lemma 5.2.3 for the relation $\sqsubseteq$ (instead of $\sim_i$).

LEMMA 5.3.2. *Let c be a computation, and let e be an event which can occur after c. For any consistent cut σ of $c \cup \{e\}$ in which e occurs, $\sigma \setminus \{e\}$ is a consistent cut of c.*

This lemma is a straightforward consequence of the equality:

$$\sigma \setminus \{e\} = \sigma \cap c.$$

PROOF OF THEOREM 5.3.1. Since $P(\phi)$ is stable, $\mathcal{A}_{P(\phi)} = \mathcal{A}^+_{P(\phi)}$. Therefore, to prove that $\mathcal{A}_{P(\phi)} \subseteq \mathcal{A}_\phi$, it is enough to show that $\mathcal{A}^+_{P(\phi)} \subseteq \mathcal{A}^+_\phi$. Let e be an event which can affirm $P(\phi)$:

$$\exists c \;:\; (c, c \cup \{e\}) \in \mathcal{C}^2 \;:\; (c \models \neg P(\phi) \text{ and } c \cup \{e\} \models P(\phi)).$$

Since ϕ implies $P(\phi)$, we have $c \models \neg\phi$. By definition of $P(\phi)$, there exists a consistent cut σ of $c \cup \{e\}$ such that $\sigma \models \phi$. Event e occurs necessarily in σ (otherwise, we would have $\sigma \sqsubseteq c$ and $\sigma \models \phi$, i.e., $c \models P(\phi)$). By Lemma 5.3.2, $\sigma \setminus \{e\}$ is a consistent cut of c. Since $P(\phi)$ does not hold at c, $\sigma \setminus \{e\} \models \neg\phi$. This shows that e affirms ϕ at σ, and so $e \in \mathcal{A}^+_\phi$.

Next, we show that $\mathcal{A}_{D(\phi)} \subseteq \mathcal{A}_\phi$. More precisely, we prove that $\mathcal{A}^+_{D(\phi)} \subseteq \mathcal{A}^+_\phi$ and $\mathcal{A}^-_{D(\phi)} \subseteq \mathcal{A}^-_\phi$.

Let e be an event of $\mathcal{A}^+_{D(\phi)}$:

$$\exists c \;:\; (c, c \cup \{e\}) \in \mathcal{C}^2 \;:\; (c \models \neg D(\phi) \text{ and } c \cup \{e\} \models D(\phi)).$$

Since ϕ implies $D(\phi)$, we have $c \models \neg\phi$. Moreover, we get an observation of $c \cup \{e\}$ by appending $\{e\}$ to any observation of c. It follows that $c \cup \{e\} \models \phi$ and thus $e \in \mathcal{A}^+_\phi$.

Consider e in $\mathcal{A}^-_{D(\phi)}$. There exists a computation c after which e can occur and such that

$$c \models D(\phi) \text{ and } c \cup \{e\} \models \neg D(\phi).$$

Then $c \cup \{e\}$ possesses an observation $\sigma_0, \cdots, \sigma_p = c \cup \{e\}$ where ϕ never holds. Let i denote the first index in $\{0, \cdots, p\}$ such that $e \in \sigma_i$. Since an observation is an increasing chain of computations, e occurs in any σ_j with $j \geq i$. Consider the sequence equal to $(\sigma_0, \cdots, \sigma_{i-1}, \sigma_{i+1} \setminus \{e\}, \cdots, \sigma_p \setminus \{e\})$, when $\sigma_i = \sigma_{i-1} \cup \{e\}$ and equal to $(\sigma_0, \cdots, \sigma_{i-1}, \sigma_i \setminus \{e\}, \cdots, \sigma_p \setminus \{e\})$, otherwise. By Lemma 5.3.2, one forms an observation of $(c \cup \{e\}) \setminus \{e\} = c$ in this way. Since $D(\phi)$ holds at c, ϕ is satisfied at least once during this observation. But ϕ cannot be observed during $\sigma_0, \cdots, \sigma_{i-1}$. Consequently, ϕ is observed after σ_{i-1}, i.e.,

$$\exists k \ : \ k \in \{i, \cdots, p\} \ : \ \sigma_k \setminus \{e\} \models \phi.$$

As ϕ does not hold at σ_k, it follows that e rejects ϕ at $\sigma_k \setminus \{e\}$, and so e belongs to $\mathcal{A}_\phi^-$. $\square$

Any computation of a traditional sequential system can be observed in a unique manner. Consequently, $P(\phi)$ and $D(\phi)$ are identical in this case. Moreover, when observing a computation of the form $c \cup \{e\}$ (where e is an event which can occur after some computation c), one first observes c and then e. This explains why Theorem 5.3.1 is obvious for traditional sequential systems.

In a distributed system, when observing a global state, one misses certain "previous" states. In this manner, it may be the case that a fact ϕ is observed during an observation of $c \cup \{e\}$ and is never observable up to c but does not hold at $c \cup \{e\}$. The inclusion $\mathcal{A}_{P(\phi)} \subseteq \mathcal{A}_\phi^+$ removes this paradox partially since it ensures that e belongs to $\mathcal{A}_\phi^+$, i.e., there exists a computation (maybe different from c) in which e makes ϕ true.

Intuitively, the inclusion $\mathcal{A}_{D(\phi)}^+ \subseteq \mathcal{A}_\phi^+$ seems quite clear. More subtle is the inclusion $\mathcal{A}_{D(\phi)}^- \subseteq \mathcal{A}_\phi^-$. Indeed, having traditional sequential systems in mind, it seems rather counterintuitive that $\mathcal{A}_{D(\phi)}^-$ is not empty, i.e., that a fact which holds in all observations of a computation c may never be satisfied when reaching a subsequent computation $c \cup \{e\}$. However, the inclusion $\mathcal{A}_{D(\phi)}^- \subseteq \mathcal{A}_\phi^-$ guarantees that such an event e is able to falsify ϕ directly.

Finally, Theorem 5.3.1 states that if an event belongs to $\mathcal{A}_{P(\phi)}$ or $\mathcal{A}_{D(\phi)}$, i.e., modifies the observation of a fact ϕ, then it can affect ϕ directly, thus indicating that the global view of a distributed system provided by its various observations is accurate enough to be useful.

Cooper and Marzullo [1991] describe an algorithm for the detection of $P(\phi)$ and $D(\phi)$. Vector time is used to determine all the possible states, i.e., all the consistent cuts. This step may be very expensive when dealing with a large number of processes (the size of vector clocks equals the number of processes) or when the computation contains many events. Concerning the second point, Cooper and Marzullo suggest that all the events which are not relevant from the viewpoint of a predicate ϕ be ignored. More formally, they consider a weakened version of vector clocks which progress only with events that potentially affect ϕ. This improvement does work because it does not forget possible global states which change the truth values of $P(\phi)$ and $D(\phi)$. A posteriori, this point is guaranteed by Theorem 5.3.1.

6. CONCLUSIONS

The results of this article demonstrate a striking analogy between local notions, defined by means of the equivalence relations $\sim_i$, and the temporal notions based on the order $\sqsubseteq$, which expresses time progress. Thus, we have shown that the predicate transformers K_i play the same role with respect to the relations $\sim_i$ as

P plays with respect to $\sqsubseteq$. Furthermore, we have proved that local predicates are exactly the fixed points of the K_i's whereas stable predicates are the fixed points of P. These results reveal how useful the ordered structure of consistent cuts is.

Our result concerning the fact that a process never forgets is a clear indication that the combinations of the predicate transformers K_i and P allow interesting properties of distributed systems to be expressed. This suggests the "instilling" of P into knowledge theory, i.e., it suggests that we consider the predicates $K_iP(\phi)$ instead of $K_i(\phi)$.

In order to give a temporal counterpart of the theorem on knowledge change, we have been led to consider the set $\mathcal{A}_\phi$ of events that potentially affect some predicate ϕ. Our results shed some light on the key role played by $\mathcal{A}_\phi$ in the study of ϕ. In particular, we have characterized local and stable predicates by means of this set. In a similar way, it would be very interesting to investigate the structure of $\mathcal{A}_\phi$ when ϕ is observer-independent. More generally, the study of the set $\mathcal{A}_\phi$ seems to be a topic worth pursuing.

ACKNOWLEDGMENTS

We would like to thank Stefan Fünfrocken, Friedemann Mattern, and Reinhard Schwarz for many helpful discussions throughout the development of this work and for a careful reading of a preliminary version of this article. We also thank Yoram Moses, Scott Stoller, and the referees whose comments have been a great help to improve the quality of this article. We are grateful to Leslie Lamport for pointing out the relation between our work and his [Lamport 1990].

REFERENCES

Chandy, K. 1988. Theorems on computations of distributed systems. Tech. Rep. 88–6, Calif. Inst. of Technology, Pasadena, California.

Chandy, K. and Lamport, L. 1985. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst. 3,* 1, 63–75.

Chandy, M. and Misra, J. 1986. How processes learn. *Distrib. Comput. 1*, 40–52.

Cooper, R. and Marzullo, K. 1991. Consistent detection of global predicates. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging.* ACM, New York, 163–173.

Dijkstra, E. and Scholten, C. 1990. *Predicate Calculus and Program Semantics.* Springer–Verlag, New York.

Fidge, C. 1988. Timestamps in message passing systems that preserve the partial ordering. In *Proceedings of the 11th Australian Computer Science Conference.* University of Queensland, Queensland, Australia, 55–66.

Halpern, J. and Moses, Y. 1990. Knowledge and common knowledge in a distributed environment. *J. ACM 37,* 3, 549–587.

Katz, S. and Peled, D. 1990. Interleaving set temporal logic. *Theor. Comput. Sci. 75*, 263–287.

Lamport, L. 1990. *win* and *sin*: Predicate transformers for concurrency. *ACM Trans. Program. Lang. Syst. 13,* 3, 396–428.

Lamport, L. 1978. Time, clocks and the ordering of events in a distributed system. *Commun. ACM 21,* 7, 558–564.

Mattern, F. 1988. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms.* North-Holland, Amsterdam, 215–226.

Nielsen, M., Plotkin, G., and Winskel, G. 1981. Petri nets, events structures, and domains, part 1. *Theor. Comput. Sci. 13*, 85–108.

PANANGADEN, P. AND TAYLOR, K. 1992. Concurrent common knowledge: Defining agreement for asynchronous systems. *Distrib. Comput. 6*, 73–93.

PLOTKIN, G. AND PRATT, V. 1990. Teams can see pomset. Available at ftp://boole.stanford.edu//pub/pp2.tex.

SANDERS, B. 1991. A predicate transformer approach to knowledge and knowledge–based protocols. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing.* ACM, New York, 217–230.

SCHWARZ, R. AND MATTERN, F. 1994. Detecting causal relationships in distributed computations: In search of the holy grail. *Distrib. Comput. 7*, 149–174.

SZPILRAJN, E. 1930. Sur l'extension de l'ordre partiel. *Fund. Math. 16*, 386–389.

Received December 1992; revised January 1994 and August 1994; accepted September 1994