A NEW APPROACH TO DETECTION OF LOCALLY INDICATIVE STABILITY

by

Nir Shavit and Nissim Francez

computer science department

The Technion, Haifa 32000, ISRAEL

" The trees that grow together in the sun,
become as one and must as one remain,
a pair that live a dozen years as one,
never my friend, can be as two again. "

Bert Leston Taylor

ABSTRACT

A new approach to derivation of detection algorithms for stable properties in distributed systems, in cases where local indicators may be found, is presented. A local indicator is a local predicate, which, when holding, indicates that the current local state is a potential component of a stable global state. "Joint" local stability implies the global one. The approach is based on a new way of viewing the computation of a distributed program and yields a solution having several important advantages over known detection algorithms for locally indicative stable properties. Unlike the case in previous algorithms, symmetry of the solution is a natural byproduct of the derivation, and the price paid for it (in terms of message complexity, restrictions on the underlaying computation, need of global knowledge ...) is very low.

1. INTRODUCTION

The problems of detecting *termination* and *deadlock* in distributed computations have been extensively studied during the last few years[1]. Chandy and Lamport in [CL 85] collected these and similar problems into a group of so called *global stable properties* to be detected. Properties in this group are *monotonic* in the sense that from the moment they first hold in a computation they continue to hold as long as the computation proceeds without external intervention. Using their snapshot algorithm, a global view of the system may repeatedly be collected, allowing to test if the required stable property holds. In [CM 85], Chandy and Misra stress the importance of local indicativity in stability detection, namely, the existence of local indicators, the conjunction of which implies global stability. Detection of locally indicative stable properties need not involve collecting a global view of the system state. Generalizing the technique of "repeated observations" of the system state used in several former detection algorithms[2], they propose a very simple paradigm to detect *global quiescence*, that is, locally indicative stability implied by the conjunction of local indicator predicates of all processes in the system, where the indicators are such that a process satisfying its local indicator predicate does not initiate communication with other processes. They further show how in some cases their proposed solution may be modified to detect quiescence when the conjunction includes only a subset of all indicators of processes in the system. Though the full paper [SF 86] deals with detection of locally indicative stability in general, we limit ourselves here to the case of

---

[1] [AR 84], [B 84], [CL 85], [CM 82], [CM 85], [CLh 82], [DFG 83], [DS 80], [ES 84], [F 80], [FR 82], [FRS 81], [M 83], [Mo 85], [R 83], [Ric 84], [SSP 85], [T 84].

[2] [AR 84], [B 84], [CL 85], [CM 85], [DFG 83], [F 80], [FR 82], [FRS 81], [M 83], [R 83], [Ric 84], [T 84].

global quiescent properties, in order to simplify the presentation, and to observe the space limitations.

In this paper, a new approach to viewing the computation of arbitrary distributed programs is presented. Based on this approach, a derivation method is presented, allowing to abandon the "repeated observations" paradigm for distributed detection, in favor of a new type of detection schemes for global quiescence, which apart from having a *very low* worst case message complexity, enjoy the following advantages:

1. They are symmetric.
2. They are not based on processes having global information (such as the number of processes or the diameter of the network).
3. They are not based on a predefined structure in the communications graph such as a cycle (Hamiltonian or other), spanning tree, etc ...
4. They are applicable in systems employing either synchronous or asynchronous communication.

After the appearance of the first detection algorithms, the issues of *symmetry* and *genericity* (see [B 85]) were raised. A number of *symmetric* algorithms for detection of global quiescent properties have been proposed. Some of the solutions [SSP 85] [ES 84] were derived on the basis of a specific type of computation behavior, and are limited to computations which are synchronized. Other solutions (e.g. [R 84], [AR 84], [Ric 84] and [B 84]) , were derived from known asymmetric algorithms (such as [FRS 82] and [CL 85]) fitting the repeated observations paradigm, their main drawback being that the performance of the derived solution is not much better, and in some aspects even worse, than the one resulting from simply running a copy of the original asymmetric algorithm from each process in the network.

The symmetric behavior and genericity of the algorithms designed using the hereby presented derivation method is a very simple and immediate result of the proposed approach. The algorithms do not suffer from the above mentioned drawbacks, and in addition have a worst case message complexity by far better than all symmetric, and of most asymmetric, previously suggested detection algorithms for global quiescent properties.

2. THE PROBLEM

2.1 Model of computation

A distributed program consists of a network of N processes and E bi-directional communication channels in the form of an undirected connected graph. Each process P_i "knows" only its own identity, and communicates (faultlessly!) with other processes by message passing along channels. A process sends messages along its incident channels in the outgoing direction and receives messages in the incoming direction, where messages sent arrive at their destination within a finite time. The *local state* of every process P_i is denoted by Y_i, and the collection of messages in transit, sent by P_i along its incident channels in the outgoing direction, and not yet received by its neighbors at the other channel ends, is denoted by the *communication state* X_i. The reason for associating such a communication state with each process will become clear in section 2.2. Message passing may be either synchronized or non-synchronized, where in the former case $X_i \equiv \phi$, $i \in \{1,..,N\}$.

From the viewpoint of an external observer, at any time, a global state $((X, Y): (X_i, Y_i), i \in \{1,..,N\})$ of the program is a collection of local and communication states (X_i, Y_i) of each and every process. The program has a set of global states, and a set of state transitions. A global state (X', Y') is said to be *reachable* from a global state (X, Y) if and only if there exists a sequence of transitions from (X, Y) to (X', Y'). A *distributed computation* is a sequence of global states, each reachable from the one preceding it. The notation $\alpha \overset{P}{\Rightarrow} \beta$ is used to denote that α implies β assuming all invariants of the distributed program P hold.

2.2 Global quiescence

A stable property of a distributed computation is a property that after holding in some global state (X, Y) will hold in any new state (X', Y') reachable from state (X, Y). It may therefore be called monotonic, in the sense that once it becomes ***true*** it will remain ***true***, (as long as no external intervention occurs). A ***global quiescent*** property $Q(X, Y)$ of a distributed computation is special type of stable property, characterized by the existence of local stability-indication predicates $S_i(Y_i)$, $i \in \{1,..,n\}$ (not necessarily monotonic!), each applicable to the local state of one process, such that

A1: (*quiescence*)

While S_i holds, a process P_i does not send messages, and as long as no messages are received, S_i continues to hold[3].

A2: (*local indicativity*)

$$\bigwedge_{i \in \{1,...,N\}} (S_i(Y_i) \wedge X_i = \phi) \quad \overset{P}{\Longleftrightarrow} \quad Q(X, Y)$$

A process P_i which is locally stable and whose priorly sent messages have all been received (i.e. satisfying $S_i(Y_i) \wedge X_i = \phi$) is said to be ***quiet*** since it cannot be "heard" by other processes (i.e. they cannot receive messages from it and therefore their local states cannot be affected by it). For ease of presentation $S_i(Y_i)$ is denoted as S_i, and P_i's being quiet is denoted by Q_i.

The difference between global quiescent properties and other quiescent properties such as "local communication deadlock", is that for global quiescent properties, the conjunction of $A2$ ranges over all processes and channels in the program, while for other non-global quiescent properties only a subset of all processes and channels are included in the conjunction.

2.3 The detection problem

Suppose that during a computation of a distributed program P, (referred to as the *basic computation*), a global quiescent property $Q(X, Y)$ is achieved. The problem is to devise another computation (***detection***) to be ***superimposed*** on P, to detect global quiescence and distribute this knowledge among all processes. In each and every process P_i, a monotonic local predicate ***detected$_i$***, which is initially ***false***, is set to ***true***, once global quiescence is detected. Correctness of the ***detection*** is expresses by means of the following two conditions:

*CC*1:

$$\bigwedge_{i \in \{1,...,N\}} (\mathit{detected}_i \overset{P}{\Longrightarrow} Q(X, Y))$$

*CC*2:

If $Q(X, Y)$ holds then eventually ***detected$_i$*** will be set to ***true*** by all processes.

2.4 Approaching the problem

Previously presented detection algorithms for quiescent properties made use of the existence of local stability-indicators to overcome the need to collect a global state of the program in order to detect stability. They preserve $CC1$, by directly maintaining

[3] Note that in some systems where communications are synchronized (such as *CSP*), communication may not be between a "sender" and a "receiver" as described above, but rather have the form of what is sometimes termed as a "handshake". For such systems $A1$ should be slightly revised to state: Two processes both satisfying S_i do not communicate, and for any process, if S_i holds, it will continue to hold as long as no communication takes place.

an invariant of the form:

IVF:

$$\alpha \overset{P}{\equiv}> \gamma$$

where α is the disjunction of all ***detected***$_i$ predicates, and γ is a conjunction of all the (***not necessarily monotonic***) local indications of a process being quiet. To do so, some algorithms have a special process (responsible for detection) maintain a trace path (defined later) to every unquiet process. Other algorithms maintain *IVF* by repeatedly testing the (non-monotonic) local stability indicators of the conjunction in some order, setting the local ***detected***$_i$ predicate of a process P_i to ***true*** if and only if the the tested conjunction of these indicators holds. Chandy and Misra in [CM 85] generalized this latter method of repeated testing to form a simple paradigm for quiescence detection.

Our approach differs from those mentioned above in that we do not insist on ***directly*** maintaining an invariant of the form *IVF*. Rather, we introduce what may be called an intermediate goal of the detection scheme, reaching a state in which a global conjunction β of local indicators holds, differing from γ in that each of its components is ***monotonic***. Such a scheme must preserve an invariant of the form:

IVFa:

$$\beta \overset{P}{\equiv}> \gamma$$

Even though, as in γ, the goal conjunction β still ranges over the local predicates of all processes, the fact that each component is independently monotonic eliminates a major problem, encountered whenever attempting to test the truth of γ, namely, the need to overcome inconsistencies while testing the truth of a conjunction of predicates that are not local to one process. The detection problem is therefore reduced to a much simpler one, having each process' local ***detected***$_i$ predicate of α be such that its truth will imply β, thus maintaining an invariant of the form

IVFb:

$$\alpha \overset{P}{\equiv}> \beta$$

It will be shown that by abandoning the method of directly maintaining *IVF* in favor of the proposed decomposition, the derivation of a detection algorithm enjoying the priorly mentioned advantages is much simplified.

3. THE SOLUTION

The derivation of a scheme that achieves our intermediate goal state, while preserving an invariant of the form *IVFa*, is based on a new approach to viewing the computation of an arbitrary distributed program. We suggest viewing an arbitrary distributed computation as if it were a collection of so called "diffusing computations". Based on this approach, we derive our general solution for detection of any given quiescent property, by a series of modifications, using as a building block an algorithm for detection of this same property for the special case of a diffusing computation.

We begin (section 3.1) by presenting a stepwise derivation of Dijkstra and Scholten's very elegant termination detection algorithm [DS 80], as a template of global quiescence detection algorithms for a diffusing computation (though other algorithms for detection of a global quiescent property in a diffusing computation, such as [CM 82], could just as well have been used). In sections 3.2 and 3.3, based on the new approach, we overcome the limitations of [DS 80], continuing the derivation of section 3.1 to create an algorithm maintaining an invariant of the form *IVFa*. Finally, we show how once *IVFa* is maintained, a very simple symmetric detection scheme can be derived by reducing the problem of maintaining *IVFb* to the already solved problem of detecting quiescence of another property in a diffusing computation.

3.1 Solution for Diffusing Computation

The following is a derivation of [DS 80] in terms of the model of section 2.1. The algorithm is presented in a manner lending itself to further generalization in later sections.

Assume that there exists in the communication graph a special "initiator" process whose channels are partially blocked, that is, allow no incoming basic messages. This matches the "environment" process having no incoming directed channels, in the original description of [DS 80]. Processes are divided into two sets , where for any process P_i, $i \in init$ if P_i is the initiator and $i \in non\_init$ otherwise. A diffusing computation is a special type of distributed computation which is started in a network of quiet non-initiator processes when the initiator process, which is initially unstable (has $\neg S_i$, and is therefore also unquiet), causes some of its locally quiet neighbor processes to become unquiet, by sending them messages. They in turn possibly cause some of their neighbors to become unquiet and so on ...

To allow quiescence detection of a diffusing computation by its initiator (which will be denoted as P_{i_0}, where $init=\{i_0\}$), an algorithm is designed to maintain an invariant of the form

IV:

$$detected_{i_0} \overset{P}{\Rightarrow} \bigwedge_{i \in \{1,\dots,N\}} Q_i .$$

To this end, a scheme of new ***reply*** signals is added to the basic communication scheme. Along any channel, one ***reply*** signal is sent in the opposite direction on account of every message received by a process. It is assumed that the initiator's channels are not blocked to ***reply*** messages in either direction. For every process P_i, define the incoming deficit along a channel as the number of basic messages received, minus the number of ***reply*** signals sent along it, and the outgoing deficit along a channel as the number of messages sent, minus the number of ***reply*** signals received along it. Define for each process P_i two variables, C_i, the sum of P_i's incoming deficits, and D_i, the sum of its outgoing deficits. In the initial state of the computation, all processes, excluding the initiator which is unstable, have $C_i=0 \wedge D_i=0 \wedge S_i$, that is, they are locally stable and are neither owed nor owe any ***reply*** signals. The predicate $C_i=0$ **and** $D_i=0$ **and** S_i is chosen as the initiator's local indication that global quiescence has been reached, allowing to set ***detected***$_i$ accordingly (**bold** operators are used in predicates that are conditions in the algorithm). Following the above choice, the invariant *IV* to be maintained becomes:

IV:

$$C_{i_0}=0 \wedge D_{i_0}=0 \wedge S_{i_0} \overset{P}{\Rightarrow} \bigwedge_{i \in \{1,\dots,N\}} Q_i .$$

It is obvious that since the initiator P_{i_0} never receives basic messages, $C_{i_0} \equiv 0$. Nevertheless, the chosen indication which includes a test if $C_{i_0}=0$ is used, since this simplifies subsequent derivation in later sections.

To maintain *IV*, a number of rules are introduced, limiting the replying of basic messages so that the initiator will not receive replies to all the messages it sent until all the non-initiator processes are quiet. The derived scheme will actually allow the initiator to "trace" the computation diffusing from it.

First note that every process P_i has a local indication that $X_i=\phi$, namely, when $D_i=0$. This follows since initially every process has $D_i=0$, the sending of a message increments D_i by 1, and since when all messages sent by a process have been replied, there cannot be any messages in transit, implying the invariance of

*IV*1:

$$\bigwedge_{i \in \{1,\dots,N\}} (D_i>0 \vee X_i=\phi).$$

It is now possible to introduce

Rule 1.

The sending of ***reply*** signals by a non-initiator process P_i is restricted to channels with a positive incoming deficit, and in addition guarded by the condition:

G_i: $(C_i>1)$ **or** $(C_i=1$ **and** $S_i)$

based on which the invariance of

IV2:

$$\bigwedge_{i \in non\_init} (C_i>0 \ \vee \ S_i),$$

is proven, implying that a non-initiator process is "traced" (formally defined in the sequel) as long as it is unstable, by some process to which it owes a ***reply*** signal.

Observe that initially $\bigwedge_{i \in non\_init} S_i$ holds. From assumption *A2*, a non-initiator process becomes locally unstable only on account of basic messages received, each such message incrementing C_i by 1. Since, by G_i, the reply decreasing C_i to 0 is sent only if S_i holds, *IV2* is invariant. Note that by **Rule 1** at most one reply is delayed when a process is unstable.

In order to assure that every non-initiator process is traced not only if it is unstable, but also if it is not quiet, the control of ***reply*** signal sending is further restricted by replacing G_i of **Rule 1** by

G_i' : $(C_i>1)$ **or** $(C_i=1$ **and** $D_i=0$ **and** $S_i)$.

Observe that initially all non-initiator processes satisfy $C_i=0 \wedge D_i=0$. By G_i', any process P_i sending the ***reply*** on account of which C_i becomes 0 has $D_i=0$, and any process P_i sending a basic message incrementing D_i by 1 must have $\neg S_i$, which by *IV2* implies $C_i>0$. Therefore the following is invariant:

IV3:

$$\bigwedge_{i \in non\_init} (C_i>0 \ \vee \ D_i=0)$$

From $IV1 \wedge IV2 \wedge IV3$ it can be deduced that:

$$\bigwedge_{i \in non\_init} (C_i>0 \ \vee \ Q_i),$$

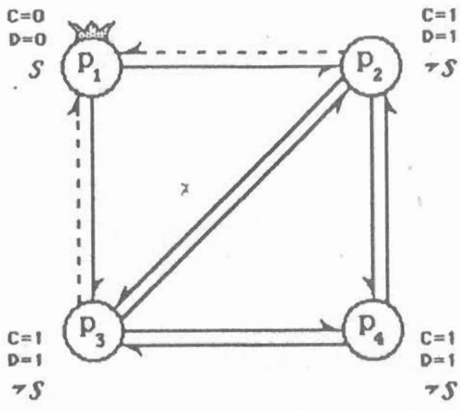
that is, every non-initiator process that is not quiet is being traced. Unfortunately, as **Example** 1 shows, the revised **Rule** 1 does not suffice to assure that the initiator's detection indication will remain unsatisfied as long as there are unquiet non-initiator processes.

Therefore ***reply*** signal sending is further controlled using a data structure which Dijkstra and Scholten dubbed a ***cornet***, ordering basic messages by the time of their receipt, and imposing that the sending of a ***reply*** signal in every process at any time, adhere to:

Rule 2.

The oldest unreplied basic message is the very last to be replied.

Based on the above, the notion of tracing is formally defined by defining a ***trace graph***, a directed graph based on the undirected communication graph, describing the tracer-traced relationships between processes. The graph consists of ***trace edges***, each pointing from a process P_i to a process P_j if and only if the last message yet to be replied by P_j, the oldest by the order maintained using P_j's cornet, was sent by P_i. For every such trace edge, P_i is said to be tracing P_j, and P_j traced by P_i.



Example 1: Circles denote processes, arrows the directions of the bidirectional channels, where dotted arrows denote the initiators incoming channels which are blocked to basic messages. Assume a diffusing computation is started by the initiator P_1, which sends a message to P_2 and becomes stable. P_2 receives the message and becomes unstable, sending a message to P_3, which sends a message to P_4, which sends a message to P_2. P_2, due to **Rule 1** sends a *reply* signal to P_1 while all processes excluding the initiator remain unstable. Once the *reply* sent by P_2 is received by the initiator, the program reaches the above state, and the initiators detection condition holds even though the computation is not quiescent.

It is proposed that IV be maintained by maintaining $IV1 \wedge IV2 \wedge IV3 \wedge IV4$, where

$IV4$: There exists a path of trace edges leading from the initiator to any non-initiator process which is being traced (has $C_i>0$).

Thus, the initiator's detection indication remains unsatisfied as long as there are unquiet non-initiator processes.

Observe that:

$O1$. By $IV3$, every tracing non-initiator process is also being traced, and therefore by definition has one incoming trace edge.

$O2$. Trace edges do not form cycles, since the initiator receives no incoming basic messages, and when a non-initiator process receives a message creating a trace edge, $C_i=0$ which by $IV3$ implies it is not tracing and therefore has no outgoing trace edge.

From the above it may be concluded that the trace graph has the form of a *directed tree* with the initiator as its root, to which all and only traced processes belong. As Dijkstra and Scholten note, its edges therefore provide the paths whose existence implies the truth of $IV4$. This completes the proof of

Theorem 1.

When the initiator satisfies $C_i=0$ **and** $D_i=0$ **and** S_i, the diffusing computation is quiescent.

In the following paragraph, proof that the quiescent property will be detected if it holds, is presented. The proof, though lacking the elegance of that presented in [DS

80], lends itself more naturally to the modifications in the following sections. Begin by proving the invariance of

IV5:

$$\bigvee_{i \in \{1,\ldots,N\}} G_i > 0 \quad \overset{P}{\equiv}> \quad \bigvee_{i \in \{1,\ldots,N\}} (G_i' \vee \neg Q_i)$$

where by definition $G_{i_0}' \equiv \mathit{false}$ in the initiator P_{i_0} since $G_{i_0} \equiv 0$. *IV5* implies that as long as some ***reply*** signal is still owed, either the computation is not yet quiescent or the signaling scheme has not terminated since some ***reply*** can still be sent. The proof of the invariance of *IV5* is as follows:

Initially, *IV5* holds since all processes satisfy $G_i=0$. A non-initiator process can have G_i become greater than 0 only after receipt of a basic message, following which, either P_i is unquiet, or G_i' is satisfied, thus maintaining *IV5*. In addition, assume that contrary to *IV5*, all processes may become quiet and have $\neg G_i'$, while some message remains unreplied, thus leaving $G_i>0$ in some process P_i (a non-initiator process). To complete the proof the above assumption will be shown to lead to a contradiction.

In order to reach a state where all processes have $\neg G_i' \wedge Q_i$, every non-initiator process P_i must have G_i' become unsatisfied, a situation possible if either:

a. G_i becomes 0 in all non-initiator processes - contradicting the above assumption.

b. In some non-initiator processes G_i becomes 1, and either D_i becomes greater than 0 or the process becomes unstable -

Since in order for the assumption to hold all processes must be quiet and therefore stable, one non-initiator process or more must have $G_i=1 \wedge D_i>0$, while all others have $G_i=0 \wedge D_i=0$. Assuming that this is the case, notice that in any process, if G_i is greater than 0 it is exactly 1, and therefore the ***reply*** signal is owed along an incoming trace edge. However, trace edges form a rooted tree, leaf processes of which have not an outgoing trace edge and therefore must have $G_i=1 \wedge D_i=0$, contradicting the last assumption.

This completes the invariance proof of *IV5*.

After the basic computation becomes quiescent, no G_i can increase. By *IV5*, the sending of reply signals decreasing the sum of all the G_is will cease only when $\bigwedge_{i \in non\_init} G_i=0$. Since this implies that all replies owed to the initiator have been sent, it may be concluded that

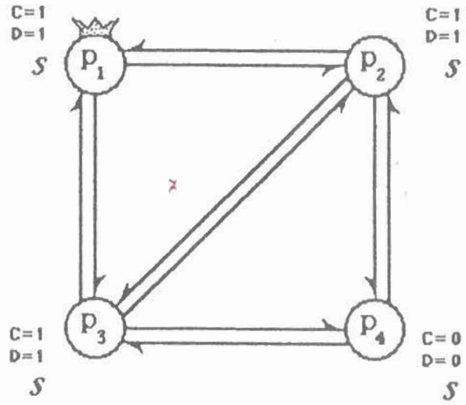
Theorem 2.

Within a finite time after the diffusing computation becomes quiescent, the initiator satisfies $G_i=0$ **and** $D_i=0$ **and** S_i.

This completes the description of Dijkstra and Scholten's detection algorithm of quiescence in diffusing computations. In the following sections we show how, based on our new approach, efficient symmetric detection algorithms of global quiescence in arbitrary distributed programs may be derived.

3.2 Towards a general solution

We begin by removing the initiator's limitation of having $G_i \equiv 0$, maintained by allowing it to receive no basic messages along its channels. To do so, the possibility of a deadlock situation falsifying *IV5* must be prevented. Such a situation can arise once the initiator is allowed to receive basic messages and return ***reply*** signals like all other processes (see **Example 2**).



Example 2: Note that in the above figure, none of the channels are blocked. P_1, the initiator, sends a message to P_2, which sends a message to P_3, which sends a message to P_1. Following this series of communications, S_i holds in all processes. However, notice that the *reply* signal scheme is deadlocked, leaving $\neg G_i'$ in all processes, while some still have $C_i > 0$. Thus $IV5$ is falsified by the creation of a deadlock cycle that includes the initiator.

To prevent the *reply* signaling scheme from deadlocking, a new local boolean flag I_i is added to each process P_i, distinguishing the initiator process from non-initiator processes by setting

$$I_i \equiv \begin{cases} true & i \in init \\ false & i \in non\_init \end{cases}$$

We call the variable I_i a *flag* to stress that its preset value remains unchanged throughout the computation. The signaling guard in all processes (including the initiator) is revised to:

$$G_i'' : (I_i \text{ and } C_i > 0) \text{ or } (\neg I_i \text{ and } (C_i > 1 \text{ or } (C_i = 1 \text{ and } D_i = 0 \text{ and } S_i))).$$

Note that based on the new G_i'', all non-initiator processes act as with G_i', while the initiator returns *reply* signals unconditionally.

However, after this modification, the former invariance proof of $IV5$ does not apply. Even though the first part of the proof holds since the initiator process P_{i_0} has G_{i_0}'' become *true* whenever C_{i_0} becomes greater than 0, the second part fails to hold because the trace graph is not necessarily in the form of a directed tree, now that the initiator may have an incoming trace edge. Notice however, that (unlike in non-initiator processes) as long as there is a trace edge incident on the initiator, $G_{i_0}'' = true$. Therefore, a situation contradicting $IV5$, that is, one in which all processes become quiet and have $\neg G_i''$, while leaving $C_i > 0$, cannot occur as long as there are trace edges incident on the initiator. It is because of this property that we redefine *trace edges* to include all trace edges fitting the former definition, excluding those pointing to an initiator process. Based on the above observation and using the new definition of a trace edge, the proof of $IV5$ again applies, implying **Theorem** 1.

Since the proofs of $IV1$, $IV2$ and $IV3$ are independent of the last change, and since by using the new definition of a trace edge, the initiator again has no incoming trace edge, and the proof of $IV4$ also applies, **Theorem 2** can also be concluded.

The revised scheme can thus be applied to allow to detect global quiescence of a diffusing computation started from any node in the communication graph.

3.3 A Collection of diffusing computations

We now proceed to derive a detection algorithm for an arbitrary distributed computation, where more than one process can satisfy $\neg S_i$ initially. We do so by making the following observation:

> An arbitrary distributed computation may be viewed as if it were a *collection*[4] of diffusing sub-computations, each having a possibly different initiator process.

Sub-computations diffusing from different initiating processes are not necessarily disjoint. For example, the sending of a message by a process P_c may depend on the prior receipt of messages from each of two processes P_a and P_b, each an initiator of a different diffusing sub-computation. In this case, the message sent by process P_c may be viewed as part of the subcomputation of P_a or P_b or both. The basis of our derivation is in allowing each initiator to trace its diffusing subcomputation, in a manner similar to that of the initiator of section 3.2. To ease the exposition and simplify the proof, we derive the general solution by continuing the derivation of sections 3.1 and 3.2. We do so by applying the revised detection scheme for a diffusing computation to an arbitrary distributed computation, attempting to maintain the invariants implying its correctness by continued modifications.

The invariance of $IV1$ is maintained, since its former proof continues to apply, being independent of processes' local stability indication. The invariance of $IV2$, though, is not maintained, since no matter which process is chosen as the initiator, many non-initiator processes may be initially unstable (satisfy $\neg S_i$) yet have $C_i=0$. To overcome this obstacle, let every initially unstable process be an initiator, that is:

$$init = \{\, i : P_i \text{ initially has } \neg S_i \,\}$$

$$non\_init = \{\, i : P_i \text{ initially has } S_i \,\}\,,$$

and set the I_i flags accordingly. Following the above revision, initially $IV2$ holds. Since by $A2$ a stable process becomes unstable only on account of a message receipt, incrementing C_i, and since by the guard G_i'' the reply decreasing C_i to 0 in a non-initiator process is sent only if S_i holds, the property $IV2$ is invariant.

Keeping in mind that following our last revision, all non-initiator processes have the flag I_i set to *false*, we note that $G_i'' \wedge \neg I_i$ equals G_i', and the former proof of $IV3$ applies. From $IV1 \wedge IV2 \wedge IV3$ it may be concluded, as before, that every non-initiator process that is not quiet, is being traced.

Since there may be more than one initiator, we weaken IV in such a way that the revised detection scheme maintains the invariance of

IV:

$$\bigwedge_{i \in init} (C_i=0 \wedge D_i=0 \wedge S_i) \overset{P}{\Longrightarrow} \bigwedge_{i \in \{1,\dots,n\}} Q_i .$$

Thus, it needs to be proven, that when all initiators satisfy $C_i=0$ **and** $D_i=0$ **and** S_i, the computation is quiescent. Using the new definition of a trace edge (i.e. one which may not point to an initiator process), we propose proving the invariance of IV by proving the invariance of $IV1 \wedge IV2 \wedge IV3 \wedge IV4'$, where $IV4'$ is a weaker form of $IV4$ such that

$IV4'$: There exists a path of trace edges leading from *some* initiator process to any non-initiator process which is being traced.

[4] A formal description of the manner in which an arbitrary distributed computation is decomposed into a collection of diffusing sub-computations can be found in the full paper, though the main idea follows from the derivation in this section.

ensuring that as long as some non-initiator is not quiet, some initiator is still tracing it as part of its diffusing sub-computation.

Based on **Rule 2** and the new definition of a trace edge we observe that $O1$ and $O2$ of section 3.1 hold. However, since there may be many initiator processes (all having no incoming trace edges), we may only come to a conclusion weaker than that of section 3.1, that the trace graph has the form of a ***forest of directed trees*** - to which all traced non-initiator processes belong - each tree having an initiator process as its root. As before, the edges of the trees in the forest provide the paths whose existence manifests $IV4'$.

The above completes the proof of the invariance of IV. However, IV is not of the desired form $IVF\alpha$, one reason being that for any initiator P_i, $C_i=0 \wedge D_i=0 \wedge S_i$ is not monotonic, and may change from ***true*** to ***false*** and back many times, following message receipts from other initiators or processes in their corresponding trace trees. A further revision is therefore needed.

Observe that once an initiator process P_i has $C_i=0 \wedge D_i=0 \wedge S_i$, it is locally stable and is actually "neutral", with respect to all other processes, as far as the detection scheme is concerned (it has ceased tracing its diffusing subcomputation and is not being traced). By $A1$ it may become unstable only on account of a basic message receipt, and is therefore, again from the viewpoint of the detection scheme, in the same state as a non-initiator process in the initial state (a slight difference resulting from the fact that since P_i has $I_i \equiv$ ***true***, by C_i'' the ***reply*** on account of this message will never be delayed). We therefore modify the scheme by allowing an initiator process that is stable and neutral to act as a non-initiator. To this end, in each process P_i, I_i is redefined to be a variable (instead of what we formerly coined a "preset flag"), initialized as before, yet controlled by the following

Rule 3.

If I_i **and** $C_i=0$ **and** $D_i=0$ **and** S_i then I_i := ***false***.

The initiator process' satisfying of $C_i=0$ **and** $D_i=0$ **and** S_i prior to application of **Rule 3** ensures the continued invariance of IV. From **Rule 3** and since by initialization all non-initiator processes satisfy $\neg I_i$, IV may be rephrased as:

IV:

$$\bigwedge_{i \in \{1 \ldots N\}} \neg I_i \overset{P}{\Longrightarrow} \bigwedge_{i \in \{1 \ldots N\}} Q_i$$

The invariance of IV is based on initializing I_i according to each process' initial condition, stable or unstable. To overcome the need to precalculate S_i for every process P_i in order to perform its initialization, we propose that initially $\bigwedge_{i \in \{1 \ldots N\}} I_i$, that is, all processes will be initiators.

It may be easily verified that this last modification does not impair the invariance proof of IV, since processes which are initially stable in the scheme with the modification, may be regarded as initiators that become stable prior to their sending or receiving any messages in the former unmodified scheme. We therefore conclude that

Theorem 3.

> When all processes have ceased being initiators, the basic computation is quiescent.

Since I_i is monotonic, IV is an invariant of the form $IVF\alpha$, and a state when all processes have ceased being initiators is the desired intermediate goal state of form β. It remains to be shown that if the computation is quiescent, the intermediate goal state will be reached within finite time.

In order to prove that $IV5$ holds for the revised scheme, one needs only to replace the observation that the trace graph is a tree of trace edges, by the observation that it is a forest, in the former proof. Since after the basic computation becomes quiescent, no C_i or D_i can increase, and since the sending of ***reply*** signals will, by *IV5*, cease only when all processes have $C_i=0$, within a finite time all processes will also have $D_i=0$. By **Rule 3**, since all processes are quiet and therefore stable,

Theorem 4.

Within a finite time after the computation becomes quiescent, all processes will cease being initiators.

This completes the derivation of a tracing scheme to reach the desired intermediate goal state, while maintaining an invariant of the form *IVFα*.

3.4 Detecting quiescence

Finally, a scheme allowing each and every process to detect global quiescence must be designed. We impose two requirements upon the scheme:

1. Processes behave symmetrically
   (This eliminates a solution in which at some point a leader is chosen).
2. Processes do not have global knowledge
   (such as the number of processes, the structure of the network ...).

To allow symmetric detection, a scheme must be added to maintain an invariant of the form:

$$\bigwedge_{i \in \{1,\ldots,N\}} (\textit{detected}_i \overset{P}{\Longrightarrow} \bigwedge_{i \in \{1,\ldots,N\}} \neg I_i)$$

One obvious solution is what may be called a notification scheme. Since in each process I_i becomes $\textit{false}$ only once (when P_i ceases being an initiator), let each process broadcast a $\textit{not-init}_i$ message when $I_i=\textit{false}$. Each process P_i will detect global quiescence upon

$$\neg I_i \textbf{ and } \{ \text{ received a } \textit{not-init}_j \text{ message for all } j \neq i \ \}.$$

Thus, within finite time all broadcasts will be completed, and all processes will "know" that global quiescence has been achieved. The major drawback of the notification scheme is the need for each process to know the total number of processes in the network in order to be able to check that a $\textit{not-init}_j$ message has been received for all $j \neq i$. This does not meet our second requirement, and therefore we pursue a different solution.

Suppose a special diffusing test computation T^j is initiated by a process P_j, achieving a quiescent property $t^j(X, Y)$, such that

$$t^j(X, Y) \overset{P}{\Longrightarrow} \bigwedge_{i \in \{1,\ldots,N\}} \neg I_i .$$

By detecting $t^j(X, Y)$, and based on *IV*, process P_j can detect $Q(X, Y)$. Since the quiescence detection algorithm for diffusing computations as presented in section 3.1 can be used to allow P_j to detect $t^j(X, Y)$, we need only devise such a diffusing test computation T^j, and a corresponding quiescent property $t^j(X, Y)$, for each process P_j. The following testing scheme T is a collection of simple broadcasts T^j, each initiated by a process P_j, $j \in \{1,..,N\}$. To distinguish between messages of different broadcasts (tests of different processes), each test message is tagged with the identification of the test's initiator. The following two rules are added to our detection algorithm:

Rule 4. (*initiation rule*)

A process P_j sends a $\textit{test}_j$ message once along each of its incident channels after it ceases being an initiator ($\neg I_j$ holds).

Rule 5. (*propagation rule*)

A process P_i, having received at least one $\textit{test}_j$ message, $j \neq i$, eventually sends a $\textit{test}_j$ message once along each of its incident channels.

By choosing $t^j(X, Y)$ to be

$$\wedge_{i \in \{1,\dots,N\}} (\neg I_i \textbf{ and } \{ P_i \textit{ received a test}_j \textit{ message along each incident channel } \})$$

we assure that

$$t^j(X, Y) \overset{P}{\equiv>} \wedge_{i \in \{1,\dots,N\}} \neg I_i .$$

By induction on the set of channels, it may be proven that within a finite number of steps after all processes have ceased being initiators, all processes will have sent and received test signals along all of their outgoing channels, and $t^j(X, Y)$ will hold.

To allow each and every process P_j to trace its corresponding query T^j independently (run a separate detection scheme) in order to detect $t^j(X, Y)$, ***reply*** messages are tagged with the initiators identification (***reply***$_j$). Each process maintains the cornet and deficit counters for tracing each query T^j (note that creation of the cornet and deficit counters can be performed for each T^j upon receipt of the first ***test***$_j$ message). Thus, by applying the proofs of section 3.1 separately to the detection scheme of each computation T^j (only ***test***$_j$ and ***reply***$_j$ signals are considered per T^j), it may be concluded that:

Theorem 5.

$$\wedge_{j \in \{1,\dots,N\}} (\{ P_j \textit{ has detected } t^j(X, Y) \} \overset{P}{\equiv>} Q(X, Y))$$

and within a finite number of steps after $Q(X, Y)$ holds,

$$\wedge_{j \in \{1,\dots,N\}} \{ P_j \textit{ has detected } t^j(X, Y) \}$$

will also hold.

4. DISCUSSION

4.1 A few comments

1. Many optimizations to both the tracing and testing schemes are possible, yet remain unmentioned to simplify the presentation.
2. The requirement that $C_i=0$ when a process becomes a non-initiator according to **Rule 3**, is added only to simplify the proof of theorems 3 and 4. Introduction of this requirement allows to redefine the notion of a trace edge, by preventing a situation where a trace edge pointing to an initiator process would become a trace edge pointing to a non-initiator, following application of **Rule 3**. Dropping this requirement will not impair the correctness or performance of the algorithm.
3. The ***tracing*** (section 3.3) and ***testing*** (section 3.4) phases need not follow one another, that is, each process may simultaneously take part in both. It might be interesting to note that from the viewpoint of any process, quiescence detection amounts to tracing two diffusing computations, one belonging to the basic computation and the other to T.

4.2 Properties of the solution

As mentioned before, other algorithms for detection of global quiescent properties in diffusing computations can be modified according to the above method. One such example is Chandy and Misra's algorithm [CM 82], which after modification will symmetricly detect termination or deadlock in *CSP* programs [H 78]. Note that the use of different schemes to acquire and distribute the knowledge that a state in which an invariant of the form β holds has been reached, results in a whole new class of quiescence detection algorithms.

Analysis of the derived algorithm's message complexity yields the following. In the tracing scheme, one ***reply*** signal is sent per basic message sent. During each process' query T^j, at most one ***test***$_j$ signal is sent by each process along all of its outgoing channels, amounting to $O(E)$, and since each process' corresponding detection scheme tracing T^j also takes $O(E)$ $reply_j$ signals, all N queries take a total $O(N^*E)$ signals. Thus, for a basic computation in which M messages were sent, the worst case message complexity of our algorithm is $O(M+N^*E)$. As a simple comparison, the worst case message complexity of *asymmetric* solutions such as the repeated snapshot algorithm of [CL 85] or the paradigm of [CM 85], is $O(M^*E)$. The message complexity of symmetric solutions such as the repeated snapshots algorithm of [B 84] is $O(M^*N^*E)$.

Unlike former solutions, the algorithms derived using the new approach are very general in the sense that they are not limited to any predefined configuration of processors, are not dependent on any form of synchronization of communications or of the basic computation, and do not relay on the existence of global information in any process.

4.3 A broader view of the approach

We see two directions in which the approach may be extended. The first is the application to detection of locally indicative stability in general[5]. The second is to use the proposed view of distributed computations as collections of diffusing sub-computations, in development and the analysis of distributed programs.

4. ACKNOWLEDGMENTS

The part of the second author was partially supported by the fund for promotion of research, the Technion. We thank Shmuel Katz and Esther Stein for their constructive criticism of initial versions of this work.

5. REFERENCES

[AR 84] Apt, K.R. and Richier, J-L., Real time clocks versus virtual clocks, Tech. Rep. 83-84, L.I.T.P., Universite Paris 7, Paris, 1984.

[B 84] Bouge, L. Repeated Synchronous Snapshots and their implementation in CSP, Tech. Rep. No. 84, L.I.T.P., Universite Paris 7, Paris, 1984.

[B 85] Bouge, L. Symmetry and Genericity for *CSP* in distributed systems, Tech. Rep. No. 85-32, L.I.T.P., Universite Paris 7, Paris, May 1985.

[CL 85] Chandy, K.M. and Lamport, L., Distributed Snapshots: Determining the global states of distributed systems, ACM Transactions on Computer Systems, vol. 4, no. 1, pp.63-75, February 1985.

[CM 82] Chandy, K.M. and Misra, J., Termination Detection of Diffusing computations in Communicating sequential processes, ACM TOPLAS 4, pp. 37-42, 1982.

[CM 85] Chandy, K.M. and Misra, J., A paradigm for detecting quiescent properties in distributed computations, Tech. Rep. 85-02, Dept. of Comp. Sci., Univ. of Texas, Austin, Jan 1985.

[CLh 82] Cohen, S. and Lehmann, D. Dynamic systems and their distributed termination, Proc. Symp. on Principals of distributed Comp., Ottawa, pp. 29-33, 1982.

[DFG 83] Dijkstra, E.W., Feijen, W.H. and van Gasteren, A.J.M., Derivation of a

[5] Locally indicative stable properties consist of properties having local stability indicators conforming to assumptions that are generalizations of A1 and A2 of section 2.2. Informally, they include properties where processes may send messages even when their local indication predicates are satisfied. In the full paper, we show how under an assumption of $fairness$ a completely $non-freezing$ algorithm for detection of such properties may be derived based on the presented approach.

termination detection algorithm for distributed computations, Inform. Processing Letters 16, pp. 217-219, 1983.

[DS 80] Dijkstra, E.W., and Scholten, C.S., Termination Detection for Diffusing Computations, Infor. Processing Letters 11, pp. 217-219, 1980.

[ES 84] Eriksen O., Skyum S., Symmetric Distributed Termination, Computer Science Dept., Aarhus Univ., Ny Munkegade, DK - 8000, Aarhus C, Denmark, 1984.

[F 80] Francez, N., Distributed Termination, ACM-TOPLAS, Vol. 2, No. 1, pp. 42-55, 1980.

[FR 82] Francez, N. and Rodeh, M., Achieving Distributed Termination without Freezing, IEEE Trans. soft. Eng. SE-8, pp. 287-292, 1982.

[FRS 81] Francez, N., Rodeh, M. and Sintzoff, M., Distributed Termination with Interval Assertions, Proc. Int. Colloc. Formalization of programming concepts, Peniscola, Spain, April 1981, Lecture Notes in Comp. Sci., Vol. 107, 1981.

[H 78] Hoare, C.A.R., Communicating Sequential Processes, CACM 21, 8, pp. 666-677, 1978.

[M 83] Misra, J., Detecting termination of distributed computations using markers, Proc. 2nd Ann. Symp. on principals of Distributed computing, Montreal, Quebec, Canada, pp. 290-294, 1983.

[Mo 85] Morgan, C., Global and logical time in distributed systems, Infor. Processing Letters 20, pp. 189-194, 1985.

[R 83] Rana, S.P., A distributed solution to the distributed termination problem, Infor. Processing Letters 17, pp. 43-46, 1983.

[Ric 84] Richier, J-L., Distributed Termination In CSP - Symmetric solution with minimal storage., TR. 84-49, L.I.T.P., Universite Paris 7, Paris, 1984

[SF 86] N. Shavit, N. Francez, A New Approach to Detection of Locally Indicative stability, Technion, Haifa, Israel, 1986.

[SSP 85] Szymanski, B., Shi, Y., Prywes, N., Synchronized Distributed Termination, Preliminary version presented in Proc. Symp. on Principals of distributed Comp., Minacki, Canada, pp. 29-33, 1985.

[T 84] Topor, R.W., Termination Detection for Distributed Computations, Infor. Processing Letters 18, pp. 33-36, 1984.