Global Virtual Time Approximation with Distributed Termination Detection Algorithms

Friedemann Mattern

Department of Computer Science, University of Saarbrücken,
Im Stadtwald 36, D 6600 Saarbrücken, Fed. Rep. Germany
Email: mattern@cs.uni–sb.de

Horst Mehl[*]

Department of Computer Science, University of Kaiserslautern,
P.O. Box 3049, D 6750 Kaiserslautern, Fed. Rep. Germany
Email: horst@informatik.uni–kl.de

Anneke A. Schoone and Gerard Tel[†]

Department of Computer Science, University of Utrecht,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
Email: anneke@cs.ruu.nl, gerard@cs.ruu.nl

Abstract

It is shown that distributed termination detection algorithms can be transformed into efficient algorithms to approximate the so-called Global Virtual Time (*GVT*) of a distributed monotonic computation. Typical instances of such computations are optimistic distributed simulations based on the time-warp principle. The transformation is exemplified for two termination detection algorithms, namely an algorithm by Dijkstra et al. and a new scheme based on the principle of “sticky flags”. The general idea of the transformation is that many termination detection algorithms (viz., one for each possible *GVT* value) run in parallel. Each algorithm determines a specific lower bound on the current *GVT* value. In a straightforward way,

[*] The work of H. Mehl is supported by the German National Science Foundation (Deutsche Forschungsgemeinschaft) under grant SPP-322671.

[†] The work of A.A. Schoone and G. Tel is supported by the ESPRIT II Basic Research Actions Program of the EC under contract no. 3075 (project ALCOM).

the possibly infinite bundle of parallel termination detection algorithms can be combined into a single distributed algorithm which computes a tight lower bound on the *GVT*.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Distributed*; C.2.4 [**Computer–Communication Networks**]: Distributed Systems—*Distributed applications*.

General Terms: Algorithms, Theory, Verification.

Additional Keywords and Phrases: Distributed Algorithms, Distributed Termination Detection, Parallel Simulation, Time Warp, *GVT*, Program Transformations, Assertions, Invariants.

1 Introduction

The purpose of this paper is twofold. First, to present a simple distributed termination detection scheme based on the paradigm of "sticky flags", and second to show how this and other distributed termination detection algorithms can be transformed into algorithms which determine a lower bound on the so-called *Global Virtual Time* (*GVT*) of distributed monotonic computations. *GVT* approximation algorithms are of great practical importance for distributed simulation systems [14], in particular for those based on the *time-warp* principle [16]. It was already noted by Jefferson [16] that *GVT* approximation algorithms are generalizations of distributed termination detection algorithms. Therefore, *GVT* approximation algorithms can be used to detect termination. In this paper we show that the converse is also true, namely that distributed termination detection algorithms can be transformed to obtain *GVT* approximation algorithms.

The paper is structured as follows. Section 2 states the distributed termination detection problem, presents a simple generic solution based on the "sticky flag" paradigm, and describes a specific instance of it for a virtual ring topology. In Section 3 the *GVT* problem and its relation with termination detection are explained. Also a general transformation from a collection of termination detection algorithms running in parallel to an algorithm for *GVT* approximation is given. Section 4 presents the *GVT* approximation algorithm that results if we apply the transformation to the algorithm from Section 2, together with its correctness proof. Next the transformation is applied to the termination detection algorithm of Dijkstra et al. Section 5 contains some further remarks and generalizations.

2 Termination Detection

Distributed termination detection is a “prototype problem” from the area of distributed computing, which is closely related to other important problems of the field, such as distributed garbage collection [33] and snapshot computation [6]. It has gained considerable interest in recent years, see for example [5, 11, 12, 20, 21, 32]. This section introduces the termination detection problem and demonstrates that straightforward solutions to it can be based on simple ideas whose correctness can be understood intuitively.

2.1 The Problem

Consider a distributed system that consists of processes $P_1, \cdots, P_n$ $(n \geq 2)$. With respect to the so-called *basic computation*, a process is always in one of two states, *passive* or *active*. The processes communicate solely by messages, which are assumed to be transmitted instantaneously. Some of these messages (but not necessarily all of them) are referred to as *activation messages*, i.e., they may render the receiver *active*. The basic computation behaves according to the following rules:

($\mathcal{R}1$) Only an active process is allowed to send activation messages.

($\mathcal{R}2$) A passive process becomes active when it receives an activation message.

($\mathcal{R}3$) At any time, a process may change from active to passive.

It is usually assumed that initially at least one process is active. If at some instant of time all processes are (simultaneously) passive, the basic computation has reached a stable state and it is said to be *terminated*. It is easily seen that when the basic computation is terminated, no activation messages will be sent any more, and all processes will remain passive forever.

Formally, we model the basic computation as follows. Each process P_i has a variable $state_i$ with values from $\{active, passive\}$. To distinguish activation messages from other messages of the basic computation, we denote messages as $\langle t, \cdots \rangle$ where t takes values from $\{active, passive\}$. Thus $\langle active, \cdots \rangle$ represents an activation message. We model the behavior of the basic computation by means of atomic actions such that rules $\mathcal{R}1$, $\mathcal{R}2$, and $\mathcal{R}3$ are obeyed. A process P_i can execute three different atomic actions: $\bar{\mathbf{I}}_i$, $\bar{\mathbf{X}}_i$, and $\bar{\mathbf{R}}_i$. (A process may execute other operations that affect only variables different from the ones mentioned here. Since this is of no concern here, such operations are not modeled by actions.) Rule $\mathcal{R}3$ corresponds to the internal action $\bar{\mathbf{I}}_i$.

$\bar{\mathbf{I}}_i$: $state_i := passive$

The transmission of a message to a process P_j is described in action $\bar{\mathbf{X}}_i$ where we have to take rule $\mathcal{R}1$ into account.

```
$\bar{\mathbf{X}}_i$: if $state_i = passive$ then $t := passive$ ;
     send $\langle t, \cdots \rangle$ to $P_j$
```

(Note that an *active* process is allowed to send $\langle passive, \cdots \rangle$-messages.) The receipt action of the basic computation reflects that a process becomes activated by the receipt of an activation message (rule $\mathcal{R}2$).

```
$\bar{\mathbf{R}}_i$: receive $\langle t, \cdots \rangle$;
     if $t = active$ then $state_i := active$
```

Since we assume instantaneous message transmission, messages can never be in transit. Therefore, we demand that corresponding send and receive actions are executed simultaneously. Otherwise, the actions of the computation are executed sequentially, and they can be executed at any time and in any order.

In this model, termination is equivalent to: $\forall i \; state_i = passive$. The *termination detection problem* consists of superimposing on a given basic computation a control algorithm which detects this global property without interfering with the underlying computation. The algorithm should satisfy the following two formal properties.

($\mathcal{T}1$) *Safety*: The algorithm does not detect termination unless the basic computation has terminated.

($\mathcal{T}2$) *Liveness*: If the basic computation has terminated then the algorithm will detect termination in finite time.

Since in a distributed system it is in general impossible to inspect the states of all processes simultaneously, termination detection is a non-trivial problem.

2.2 A Simple Termination Detection Principle

For a first attempt to solve the problem, assume that each process P_i has a *state indicator* S_i reflecting the state of the process (i.e., $S_i = state_i$). Then an initiator may start a *control wave* which visits all processes and returns the values of the state indicators. (More efficiently, it could only return the "accumulated" value *passive* if all processes were passive, and *active* otherwise.) To model algorithms that implement control waves the concept of *total algorithms* was introduced in [31]. Such algorithms can be implemented in various ways; examples include a control message circulating on a (virtual) ring connecting all processes (see also Raynal and Hélary [25]), parallel distributed graph traversal schemes such as the echo algorithm [8], and (virtual) broadcast schemes on a spanning tree.

Unfortunately, however, the values of the state indicators collected in that way do not allow the conclusion that the basic computation has terminated. Because of possible reactivations of processes "behind the back" of the wave, the observation that all processes were passive when being inspected (by the wave) does not imply that all processes were passive simultaneously. (Notice that *control* messages are not assumed to be transmitted instantaneously; however, even for instantaneous control messages the complete execution of the control wave algorithm is not instantaneous because the wave may be delayed at processes it inspects. Therefore, processes may be reactivated while the wave is in progress.) An algorithm that announces global termination when it finds that all S_i were passive may erroneously detect termination.

Sticky Flags. Fortunately, the simple scheme sketched above can easily be transformed into a correct algorithm. Assume now that the state indicators S_i are "sticky" in the following sense. If a process P_i is activated, the value of S_i becomes (or remains) active. If a process becomes passive, however, S_i "sticks" to active. Before the start of the termination detection algorithm, the state indicators of all processes are initialized to the value of $state_i$, thus correctly reflecting the state.

Formally, we need to augment the receipt action $\bar{\mathbf{R}}_i$ of the basic computation with the proper assignment to S_i. The receipt action $\bar{\mathbf{R}}_i$ becomes:

```
$\bar{\mathbf{R}}_i$:  **receive** $\langle t, \cdots \rangle$;
      **if** $t = active$ **then begin** $state_i := active$; $S_i := state_i$ **end**
```

As S_i is *not* set to *passive* when the state of the process becomes *passive*, the internal action $\bar{\mathbf{I}}_i$ is not changed, nor is action $\bar{\mathbf{X}}_i$.

Clearly, if at the start of the control wave some process P_j was active, the algorithm will not announce global termination because the value of S_j is still *active* when it is eventually collected by the wave. Or, to put it in another way: If the algorithm reports termination, then no process was active at the start of the wave; hence the basic computation has actually terminated because it was already terminated when the wave was started. This shows that the implicit semantics of the sticky state indicators ensures the *safety* of the resulting termination detection algorithm.

Unfortunately, however, in the scheme as it stands termination will never be announced unless all processes were initially passive. To guarantee *liveness* it is necessary to repeatedly first reset the sticky state indicators to the true values of their processes' states and then start a new control wave. Then, when the basic computation terminates, eventually the sticky state indicators will be set to *passive* (and never reset to *active*). Consequently, termination will be announced at the end of the next wave. In order not to compromise the safety property however, a state indicator must not be reset to *passive* between the start of a wave and the collection of its value.

A concrete instance of the "sticky-flag" scheme using a circulating control message will be shown in the next section; a formal proof of the liveness and the safety property for a generalized variant of the "sticky flag" scheme will be given in Section 4.2.

2.3 A Termination Detection Algorithm

We now present a concrete instance of the general scheme described above. For the superimposed termination detection algorithm we assume that the processes $P_1, \cdots, P_n$ ($n \geq 2$) are arranged to form a logical ring on which a control message circulates. (Recall that messages of the basic computation can be sent from any process to any other process.) The termination detection algorithm makes use of the variable S_i, the sticky state indicator, which reflects whether process P_i has been *active* since the last visit of the control wave. A dedicated process, P_n, initiates the algorithm by sending a control message to the next process (i.e., P_1) on the ring:

```
send ⟨$passive$⟩ to $P_1$
```

When receiving the circulating control message, process P_i executes action $\bar{\mathbf{W}}_i$ atomically.

```
$\bar{\mathbf{W}}_i$:  (1)  receive ⟨$M$⟩;
       (2)  if $S_i = active$ then $M := active$;
       (3)  if $i = n$ and $M = passive$ then signal termination;
       (4)  if $i \neq n$ then send ⟨$M$⟩ to $P_{i+1}$
       (5)           else  send ⟨$passive$⟩ to $P_1$;
       (6)  $S_i := state_i$
```

In line (1) the contents of the received message is assigned to M. In line (2) M accumulates the value of the state indicator S_i. If after a complete round M is still *passive* (3), termination can be signaled according to the arguments of Section 2.2. In any case the control message is propagated; at P_n, however, it must be reinitialized to *passive* (4, 5). In line (6) the state indicator is reset to the current value of the system variable $state_i$ as described in Section 2.2.

We deliberately dispense with two modifications that would make the algorithm more efficient. First, instead of stopping the algorithm once termination has been established, the control message continues to circulate. Second, the control message is not deferred in active processes (i.e., control messages are propagated in a *non–lazy* way). In fact, the decision of a process when to propagate the control message is independent of its state. This property becomes important when the algorithm is used for GVT approximation in Section 4.1.

The algorithm we just described is reminiscent of the well-known termination detection algorithm by Dijkstra et al. [11]. However, whereas in that algorithm a flag is set when an activation message is *sent*, our scheme uses a flag (the sticky state indicator) which is set when an activation message is *received*. We shall come back to Dijkstra's algorithm in Section 4.3.

2.4 Variants

Telepathic Computations. Interestingly, the termination detection principle derived above works independently of the mechanism by which active processes reactivate passive processes. As far as termination detection is concerned, reactivations may as well be caused by some sort of "telepathy", rather than by the explicit exchange of activation messages. To model telepathic computations, we dispense with rule $\mathcal{R}1$ (i.e., we do not model the sending of messages) and replace rule $\mathcal{R}2$ by the following "telepathic reactivation rule".

($\mathcal{R}2'$) A passive process may only become active if there exists another process which is active at that moment.

Of course, due to the lack of global time and common state in distributed systems, the observance of rule $\mathcal{R}2'$ by the basic computation requires some hidden mechanism using messages. The point is, however, that the "activator" is not aware that it activates another process. Therefore, in contrast to virtually all other known termination detection algorithms, the "sticky flag" algorithm does not need to consider the messages and the send actions $\bar{\mathbf{X}}_i$ of the underlying basic computation—the only thing it has to do is to take notice of the fact that a process becomes active.

It should be noted that termination of "telepathic computations" is not a stable property in the sense of Chandy and Lamport [6] or Lai and Yang [17][1]. Nevertheless, once an instant of time has been reached where all processes are simultaneously passive (i.e., the computation has terminated), the processes remain passive. Our algorithm can be used to detect this global termination property of "telepathic computations" as the following argument shows. Whenever some process P_i becomes active according to rule $\mathcal{R}2'$, there exists another active process P_j. Conceptually, it can be assumed that P_j sent an activation message to P_i which reactivated P_i. Hence, rules $\mathcal{R}1$ and $\mathcal{R}2$ are observed. Since nothing has to be done when sending a (conceptual) message, the "sticky flag" algorithm can directly be applied on underlying "telepathic computations". This is not the case for most other termination detection algorithms[2].

Synchronous Communication. Instantaneous message transmissions are not realizable in practice, they are merely an idealization of *synchronous communication* [9]. Interestingly, however, the "sticky flag" principle can also be used with

[1] Since messages do not necessarily exist, it is easy to construct a (consistent) cut where all processes are passive and a later cut where one or more processes are active. This is due to the fact that interference between processes by another mechanism than message passing is not considered. Thus one should redefine the concept of "consistent cut" and hence of "stable property" for telepathic computations.

[2] For example, the algorithm of Dijkstra et al. [11] requires some control activity whenever a process sends a message, namely coloring the sending process black. In other schemes, activation messages must be counted [20] or acknowledged [12].

the more realistic synchronous communication mode, where the send operation blocks until the sender knows that the receiver is also blocked and ready to accept the message. Because the receiver is blocked while the activation message is in transit (which disables the visit of the receiver by a control wave), one may define the receiver to be active already at the moment the message is sent (instead of being activated when the message is actually received). Since by rule $\mathcal{R}1$ the sender is also active at the moment of sending the message, rule $\mathcal{R}2'$ is observed.

Asynchronous Communication. The "sticky flag" scheme can also be adapted to *asynchronous communication.* (In that case, the basic computation is considered to be terminated if all processes are passive and no activation messages are in transit.) One possibility to detect termination in the asynchronous case is to acknowledge each activation message and to consider a process to be engaged in a send operation (and hence to remain active) until the acknowledgement is received. Obviously, this can be realized by locally counting sent messages and received acknowledgements and by keeping S_i *active* while there are outstanding acknowledgements for process P_i. Then, again, rule $\mathcal{R}2'$ is observed. It is also possible to use indirect acknowledgements and to batch acknowledgements; these techniques are used, for example, in the vector counter algorithm [20].

3 Global Virtual Time and its Approximation

In this section, a particular distributed computation scheme is considered, which defines a monotonic function of the global state. A simplifying assumption is made about the domain of this function, namely, that it is a set of real numbers. The Global Virtual Time approximation problem consists of computing a suitable approximation of this function, as to be defined in this section. A treatment of the generalized problem, in which the domain of the monotonic function is an arbitrary partially ordered set, is given in [22, 29, 30, 32].

After defining the GVT problem, we show how termination detection and global virtual time approximation are related. This relation then leads to a general transformation of termination detection algorithms to GVT approximation algorithms.

3.1 The Global Virtual Time Problem

Distributed Monotonic Computations. Consider a system of n processes $P_1, \cdots, P_n$ $(n \geq 2)$. Each process P_i maintains a real–valued variable C_i, referred to as the *clock* of P_i. The processes interact by exchanging timestamped messages. (Again, message transmission is assumed to be instantaneous; basically the same arguments hold for synchronous communication, however. Asynchronous communication will be discussed in Section 5.1.)

The timestamps of the messages and the modification of the clocks satisfy the following rules.

($\mathcal{S}1$) The timestamp of a message is at least the clock value of the sender (at the moment the message is sent).

($\mathcal{S}2$) On receipt of a message with timestamp t the receiver's clock is set to $C_i := \min(C_i, t)$.

($\mathcal{S}3$) At any time, a process can increase its clock.

Computations that behave according to these rules are called *distributed monotonic computations* [22] because the global minimum of all clocks C_i increases monotonically during the computation. This global minimum is referred to as the *Global Virtual Time* (GVT) of the computation, see Jefferson[3] [16]. Typical instances of distributed monotonic computations are parallel discrete event simulation systems [14] where local simulator processes cooperate by scheduling so-called remote events using timestamped messages.

Rules $\mathcal{S}1$–$\mathcal{S}3$ translate to the following three atomic actions to model the behavior of a distributed monotonic computation. According to rule $\mathcal{S}3$, a process can increase its clock by an internal action $\mathbf{I}_i$:

```
$\mathbf{I}_i$:  choose $d > 0$;
     $C_i := C_i + d$
```

The transmission of a message is governed by rule $\mathcal{S}1$:

```
$\mathbf{X}_i$:  choose $t \geq C_i$ ;
     send $\langle t, \cdots \rangle$ to $P_j$
```

The receipt of a message can cause a process P_i to set back its clock. This is modeled by executing $C_i := \min(C_i, t)$ after receipt of a message with timestamp t (rule $\mathcal{S}2$).

```
$\mathbf{R}_i$:  receive $\langle t, \cdots \rangle$;
     $C_i := \min(C_i, t)$
```

The Global Virtual Time is defined for each global state of the system by the relation $GVT = \min_i C_i$. It will first be shown that GVT is indeed a non–decreasing function.

Theorem 3.1 *GVT is monotonically non–decreasing, that is, if it is changed, it is increased.*

[3] Jefferson's original definition allows in–transit messages, see also Section 5.1.

Proof. Computations of the system are modeled as sequences of atomic actions, so it suffices to show that GVT does not decrease as the result of an atomic action.

An internal action $\mathbf{I}_i$ increases C_i, possibly increasing, but never decreasing GVT. As message transmission is instantaneous, a receipt action $\mathbf{R}_i$ always corresponds to a send action $\mathbf{X}_j$ for some j (process P_j sends a message to P_i). For the timestamp t of the message sent $t \geq C_j$ holds, and $\min(C_i, t)$ is assigned to C_i. Thus $\min(C_i, C_j)$ is not decreased, and neither is GVT. □

GVT Approximation. The GVT approximation problem consists of superimposing on the distributed monotonic computation a control algorithm which maintains a suitable approximation of GVT without interfering with the basic computation. The approximation should satisfy the following two formal properties.

($\mathcal{G}1$) *Safety*: The approximation never exceeds GVT.

($\mathcal{G}2$) *Liveness*: If GVT reaches a value $x \in \mathbb{R}$ then within finite time the approximation, say, G satisfies and continues to satisfy $G \geq x$.

GVT is a function of the global state of the system, and since the global state is not directly observable by a process, GVT approximation is a non–trivial problem.

The determination of a tight lower bound on the current GVT value is of great importance for distributed simulation systems, see, e.g., Fujimoto [14] and Jefferson [16]. Since in optimistic distributed simulations a simulator process has to roll back to an earlier state when a message with an earlier timestamp than its current clock value arrives, it must save its state regularly. The GVT is the earliest virtual time to which any simulator process can ever roll back. Therefore, all checkpoints (possibly except the most recent one) older than the GVT approximation can be removed to save memory. The liveness of the approximation ensures that eventually all oblivious states (possibly except the last one) are discarded. A lower bound on GVT is also necessary to know when irrevocable actions (e.g., simulation animation or display of statistical results) can be committed.

3.2 The Relation between GVT and Termination

Termination Expressed as Global Virtual Time. It has already been observed by Jefferson and others [16, 29, 32] that distributed termination detection is a special case of GVT approximation. To see this, model an arbitrary distributed computation (cf. the rules in Subsection 2.1) as a monotonic computation, where the values of the clocks are restricted to two arbitrary values, denoted by *active* and *passive*, ordered as *active* < *passive*. Activation messages are timestamped with *active*, other messages with *passive*.

As the processes originally satisfy rules $\mathcal{R}1$–$\mathcal{R}3$, under this transformation they observe rules $\mathcal{S}1$–$\mathcal{S}3$: A process whose clock value is *passive* only sends messages with timestamp *passive.* A process sets back its clock to *active* when a message with timestamp *active* is received. When a process becomes passive, it advances its clock from *active* to *passive.* It is now the case that $GVT = passive$ means that all processes are passive—hence, the computation has terminated. Thus, any GVT approximation algorithm can be used as a distributed termination detection algorithm—when the approximated GVT reaches the value *passive*, termination can be concluded.

Termination and a Lower Bound for GVT. Conversely, however, it is also possible to check for a general distributed monotonic computation whether GVT has reached some threshold value t by using a termination detection algorithm [29, 32]. For that purpose, fix $t \in \mathbb{R}$ and divide $\mathbb{R}$ in two intervals $[\Leftrightarrow\infty, t)$ and $[t, \infty]$, and call the first *t–active* and the second *t–passive.* Then, as above, we can consider a distributed monotonic computation as a basic computation for (t)–termination, whereby t–termination is equivalent to $GVT \in [t, \infty]$. Formally, we have the following correspondence.

Theorem 3.2 *Let $t \in \mathbb{R}$. Define the pseudo-variable $state_i^{(t)}$ to have the value t–passive if $C_i \geq t$ and t–active if $C_i < t$. Define the timestamp x of a message $\langle x, \cdots \rangle$ to be t–passive if $x \geq t$ and t–active if $x < t$. Define t–terminated as $state_i^{(t)}$ = t–passive for all i.*

Then a distributed monotonic computation (modeled by $\mathbf{I}_i$, $\mathbf{X}_i$, and $\mathbf{R}_i$) can be considered as a basic computation for t–termination (modeled by $\bar{\mathbf{I}}_i$, $\bar{\mathbf{X}}_i$, and $\bar{\mathbf{R}}_i$).

Proof. We show that for every $t \in \mathbb{R}$ rules $\mathcal{R}1$–$\mathcal{R}3$ are observed. Consider atomic action $\mathbf{I}_i$. As the value of d with which C_i is increased is positive, the value of $state_i^{(t)}$ can change from *t–active* to *t–passive*, namely if $C_i < t \leq C_i + d$ for the old value of C_i. In all other cases $state_i^{(t)}$ remains the same. ($\mathcal{R}3$)

Consider action $\mathbf{X}_i$. For the timestamp x of the message transmitted we have $x \geq C_i$. Thus it is a t–activation message for those t with $t > x$. Since $t > C_i$ for those t, t–activation messages are only sent by processes for which $state_i^{(t)} =$ *t–active.* ($\mathcal{R}1$)

Consider action $\mathbf{R}_i$. Upon receipt of a message with timestamp x, C_i is set to $\min(C_i, x)$. For those values of t for which x is *t–active* (i.e., $t > x$), C_i is set to a value $< t$, hence $state_i^{(t)}$ is set to *t–active.* ($\mathcal{R}2$) □

Notice that if a computation (a process) is t–terminated (*t–passive*), then it is also t'–terminated (*t′–passive*) for all $t' \leq t$. As a distributed monotonic computation behaves according to rules $\mathcal{R}1$–$\mathcal{R}3$, t–termination is a *stable property* (i.e., once a computation is t–terminated it remains t–terminated) and a termination detection algorithm can be applied to detect t–termination. The crucial point for GVT approximation is stated in the following theorem.

Theorem 3.3 *A distributed monotonic computation is t–terminated if and only if $GVT \geq t$.*

Proof. Consider the following equivalences. t–terminated $\Leftrightarrow \forall i \;\; state_i^{(t)} =$ *t–passive* $\Leftrightarrow \forall i \; C_i \geq t \Leftrightarrow \; \min_i C_i \geq t \Leftrightarrow \; GVT \geq t$. □

The safety and liveness properties of the (t–)termination detection algorithm imply the following.

1. The algorithm does not detect t–termination unless $GVT \geq t$.

2. If $GVT \geq t$ then t–termination will be detected in finite time.

The basic idea of our GVT approximation scheme is to run many t–termination detection algorithms for different values of t in parallel. The approximation is chosen to be the largest value for which t–termination is detected. The two properties listed above ensure the safety and progress of the resulting approximation, provided that t–termination detection algorithms for an appropriate set of t–values are used. Fortunately, it is possible to simulate the parallel execution of many termination detection algorithms by a single algorithm. In the combined algorithm a control message (of finite length) represents the control messages of a possibly infinite number of virtual termination detection algorithms. This idea will be worked out in more detail in the next section.

3.3 GVT Approximation with Termination Detection

In this section it is shown that a GVT approximation algorithm can be obtained from a termination detection algorithm. The resulting transformation will be exemplified further down in Section 4 by applying it to the “sticky flag” algorithm presented earlier.

The General Transformation. The GVT approximation algorithm consists of parallel invocations of a termination detection algorithm A, where the invocation $A^{(t)}$ is responsible for the detection of t–termination. The collection of invocations of A is referred to as a *bundle* of termination detection algorithms. An approximation of GVT is held in a variable G such that the safety property ($\mathcal{G}1$) $GVT \geq G$ is maintained.

As was noted in Section 3.2, the detection of t–termination implies that $GVT \geq t$. Hence, the assignment $G := t$ can safely be executed when t–termination is detected. It is possible, however, that at a later time t'–termination is detected, for $t' < t$. In that case this assignment would set G back, and the liveness requirement ($\mathcal{G}2$) would not be satisfied. In order to obtain a monotonic approximation and avoid the problem with the liveness sketched, we will use the assignment $G := \max(G, t)$ instead of $G := t$.

The approximation algorithm is obtained by transforming a termination detection algorithm A as follows.

- Consider an invocation $A^{(k)}$ of A for every $k \in \mathbb{R}$, responsible for detecting k–termination.
- Replace the detection of termination in $A^{(k)}$ by the statement $G := \max(G, k)$.
- Execute all invocations in parallel with the basic computation.

Note that the approximation algorithm is composed of an infinite number of parallel algorithm invocations. It must be assumed for a while that it is possible to execute all these invocations in parallel in such a way that progress is made in each of them. It will be shown later how the combined algorithm can be transformed to a finite algorithm.

Theorem 3.4 *The resulting GVT approximation algorithm satisfies safety and liveness.*

Proof. To show the safety, consider the value of GVT in some system state. By Theorem 3.3, k-termination holds for all $k \leq GVT$, but for no $k > GVT$. Hence, by the safety of each $A^{(k)}$, k–termination could only be detected for $k \leq GVT$ and thus by our transformation $G := \max(G, k)$ was only executed for values $k \leq GVT$. Consequently, $G \leq GVT$ holds.

To show the liveness, assume that at some time $GVT = k$. This implies that k–termination holds, and hence, as the bundle includes an invocation $A^{(k)}$, k–termination will be detected in finite time and the assignment $G := \max(G, k)$ will be executed. After the assignment, $G \geq k$ will continue to hold. □

We shall now discuss briefly whether the liveness can also be obtained using a bundle which does not contain an invocation $A^{(k)}$ for each value of k.

Assume that the bundle does *not* include $A^{(k)}$, and neither does it include an $A^{(k')}$ for $k \Leftrightarrow \epsilon < k' < k$, for some $\epsilon > 0$. Furthermore, assume that $GVT = k$ at some time, but GVT does not grow beyond k. In this case, t–termination holds for all $t \leq k$, but for no $t > k$. As a result of the liveness of A, $G := \max(G, t)$ will be executed for all $t < k$ for which an invocation $A^{(t)}$ is present, and as a result of its safety it will not be executed for any $t > k$. Because of the "ϵ interval gap" in the bundle, it follows that $G := \max(G, t)$ will not be executed for any $t > k \Leftrightarrow \epsilon$, hence $G \leq k \Leftrightarrow \epsilon$ continues to hold. In this case the liveness requirement is not satisfied.

It remains to study the case where the bundle does *not* include $A^{(k)}$, but does include an $A^{(k')}$ for $k \Leftrightarrow \epsilon < k' < k$, for every $\epsilon > 0$. Again assume that $GVT = k$

at some time and GVT does not grow beyond this value. For every $\epsilon > 0$ there is a k' with $k \Leftrightarrow \epsilon < k' < k$ for which $G := \max(G, k')$ will be executed in finite time, and hence for every $\epsilon > 0$, $G \geq k \Leftrightarrow \epsilon$ will hold in finite time. This, however, does not imply that $G \geq k$ holds within finite time, because, for example, $G \geq k \Leftrightarrow 1/n$ may hold only after n time units.

The conclusion of this discussion is that the liveness of the resulting algorithm cannot be guaranteed if the bundle does not include $A^{(k)}$ for *every* $k \in \mathbb{R}$ which is a possible value of the GVT. In the general case this implies that an invocation $A^{(k)}$ must be included for *every* $k \in \mathbb{R}$.

Practicability. Although the derived algorithm satisfies the formal correctness requirements of a GVT approximation algorithm, its implementation still faces a problem. In practice it is not possible to execute an infinite (and even uncountable) number of separate algorithm invocations concurrently.

There may be special applications where it is not necessary to execute all invocations. For example, in computations where it is known that the GVT only takes values of a finite set $\{k_1, \cdots, k_m\}$ it suffices to execute $A^{(k_1)}, \cdots, A^{(k_m)}$. Another example includes situations where a Δ–*approximation* ($\Delta > 0$) suffices; the liveness requirement is replaced by the weaker requirement that if the GVT is at least k, then the approximation is at least $k \Leftrightarrow \Delta$ in finite time. Under this requirement it suffices to execute the invocations related to multiples of Δ; if it is known in addition that the GVT will take values only from a finite length interval, it again suffices to execute a finite number of invocations.

The feasibility of the scheme for the general case, however, depends largely on our ability to combine the steps of an infinite bundle into a finite number of steps. Fortunately, this is indeed possible for the termination detection algorithms considered here.

A first concern to manage the computational complexity is to prevent the sending of an infinite number of messages. A bundle of algorithms is called *coherent* if each of its invocations exhibits the same pattern of message exchanges. That is, although the message contents may be different, the decision whether a message is sent should be independent of the state of a process. Obviously, the non–lazy control message propagation principle (see Section 2.3) is the key property for obtaining a coherent bundle of termination detection algorithms. For such a bundle it is then (at least theoretically) possible to assume that the messages of the different invocations are *combined* into one single message (possibly of infinite length which will subsequently be reduced to finite length), carrying the information of the messages of all invocations.

Upon receipt of such a combined message by a process P_i, all invocations become simultaneously activated in P_i. (A next concern will indeed be to combine this local activity in a similar manner.) Because of the coherence of the bundle, all invocations will generate the same pattern of messages in response to the receipt.

Consequently, these messages can also be combined into a single message. Thus the total number of messages sent by the combined algorithm is the same as the number of messages sent by a single execution of A.

The next concern will be to bound the storage used in a process. Consider the pseudo-variable $state_i^{(k)}$ used by $A^{(k)}$ in process P_i. Because $state_i^{(k)} = passive$ for all $k \leq C_i$, and *active* else (see Theorem 3.2) the infinite collection of variables $state_i^{(k)}$ is succinctly represented by the "boundary value" C_i. It turns out that the same can be done for all variables of the termination detection algorithm of Section 2.3, and also for the information transmitted in the control message. This will be demonstrated in the next section, where the transformation is carried out in detail.

Not only for the algorithm of Section 2.3, but for arbitrary termination detection algorithms a transformation of the infinite bundle to a finite algorithm can be carried out. For example, in Section 4.3 we do it for the algorithm of Dijkstra et al. [11], and it is done in [29] for the case of the vector counting algorithm [20]. In the general case, for each variable v_i of the termination detection algorithm, the real line is always partitioned into a *finite* number of intervals, such that $v_i^{(k)}$ is constant for all k of an interval. (The boundaries of these intervals are determined by the timestamps which actually occur in the computation.) In that case, the infinite collection of variables $v_i^{(k)}$ can be finitely represented by maintaining the intervals and the value of v_i for each interval, as is also proposed in [29].

4 Two Simple GVT Approximation Algorithms

In this section the transformation described in Section 3.3 will be applied to the "sticky flag" termination detection algorithm of Section 2.3. The resulting finite and elegant *GVT* approximation algorithm turns out to be an already known algorithm—it was originally proposed by Tel [30] and was recently reinvented by Baldwin et al. [2]. We will rigorously prove its correctness, thereby also obtaining a correctness proof for the "sticky flag" termination detection algorithm as a special case. We conclude the section with the transformation of the DFG–algorithm by Dijkstra et al. [11].

4.1 Transformation of the Sticky Flag Algorithm

For convenience, the text of the termination detection algorithm presented in Section 2.3 is first repeated here. Process P_n initiates the algorithm by sending a control message $\langle passive \rangle$ to the next process (i.e., P_1) on the ring. When receiving the circulating control message, process P_i executes action $\bar{\mathbf{W}}_i$ atomically.

```
$\bar{\mathbf{W}}_i$:  (1)  receive $\langle M \rangle$;
       (2)  if $S_i = active$ then $M := active$;
       (3)  if $i = n$ and $M = passive$ then signal termination;
       (4)  if $i \neq n$ then send $\langle M \rangle$ to $P_{i+1}$
       (5)              else  send $\langle passive \rangle$ to $P_1$;
       (6)  $S_i := state_i$
```

Forming the Infinite Bundle. In a first transformation step an invocation $A^{(k)}$ of this algorithm is formed for each $k \in \mathbb{R}$. This invocation is responsible for detecting k–termination. Each algorithm $A^{(k)}$ has its own instances of local variables $M^{(k)}$, $S_i^{(k)}$, and conceptually, $state_i^{(k)}$. Instead of reporting k–termination, a shared variable G is set to $\max(G, k)$ in process P_n, as explained in Section 3.3. Since $C_i = k$ represents the boundary between k–*active* and k–*passive*, the statement "$S_i^{(k)} := state_i^{(k)}$" in line (6) can be replaced by "**if** $C_i \geq k$ **then** $S_i^{(k)} := passive$ **else** $S_i^{(k)} := active$". The atomic action executed by P_i of the resulting algorithm $A^{(k)}$ upon a visit of the control message is shown below. As described in Section 3.3, this is already a *GVT* approximation algorithm if the bundle of invocations $\{A^{(k)}\}_{k \in \mathbb{R}}$ is run in parallel (and terminates within finite time).

```
(1)  receive $\langle M^{(k)} \rangle$;
(2)  if $S_i^{(k)} = active$ then $M^{(k)} := active$ ;
(3)  if $i = n$ and $M^{(k)} = passive$ then $G := \max(G, k)$;
(4)  if $i \neq n$ then send $\langle M^{(k)} \rangle$ to $P_{i+1}$
(5)              else  send $\langle passive \rangle$ to $P_1$;
(6)  if $C_i \geq k$ then $S_i^{(k)} := passive$ else $S_i^{(k)} := active$
```

Merge to a Single Algorithm. In the second transformation step, a *single* algorithm is obtained which simulates the whole bundle. This transformation employs the non–laziness of the termination detection algorithms thereby enabling a coherent execution of the bundle. The basic idea is to combine the infinite number of control messages to a single (infinitely long) control message $\langle \{M^{(k)}\}_{k \in \mathbb{R}} \rangle$. Process P_n initiates the algorithm which simulates the entire bundle by sending $\langle \{passive\}_{k \in \mathbb{R}} \rangle$ (i.e., a control message that contains the value *passive* for each $k \in \mathbb{R}$) to P_1. When receiving the circulating control message, P_i executes the following action atomically.

```
(1)  receive $\langle \{M^{(k)}\}_{k \in \mathbb{R}} \rangle$;
     forall $k \in \mathbb{R}$ do
(2)          if $S_i^{(k)} = active$ then $M^{(k)} := active$ ;
(3)          if $i = n$ and $M^{(k)} = passive$ then $G := \max(G, k)$
     enddo;
```

```
(4)   if $i \neq n$ then send $\langle\{M^{(k)}\}_{k\in\mathbb{R}}\rangle$ to $P_{i+1}$
(5)           else send $\langle\{passive\}_{k\in\mathbb{R}}\rangle$ to $P_1$;
      forall $k\in \mathbb{R}$ do
(6)           if $C_i \geq k$ then $S_i^{(k)} := passive$ else $S_i^{(k)} := active$
      enddo
```

The statements in the **forall** loops can be thought of as being executed for each k in parallel.

Confining to Finite Resources. For the next step in the derivation (the combination of the infinite number of two–valued variables into one real–valued variable) we need the following theorem. It states that for each variable of the algorithm the infinite bundle of boolean variables can be represented by a single real–valued variable (which may include the values ∞ and $\Leftrightarrow\infty$). The proof of the theorem implies how these real–valued variables must be updated under the operations of the bundle of algorithms.

In proofs of theorems about distributed algorithms we use the method of "system–wide invariants" [28]. The idea is to express the desired (safety) property of an algorithm as an assertion about values of program variables and to prove its correctness by means of invariants. An invariant is an assertion with the following properties: (1) The assertion holds initially (i.e., for the initial values of the program variables before any action is executed), and (2) for every atomic action we have the following: Assuming the assertion holds before, it also holds after the execution of the action. It is clear from this definition that an assertion which is an invariant is always true during any execution of the system. Hence this is an orderly way to formally prove properties of distributed algorithms.

Theorem 4.1 *(a) At any time for each process P_i there exists a real number s_i such that for each k, $S_i^{(k)} = active$ for $k > s_i$ and $S_i^{(k)} = passive$ for $k \leq s_i$.*
(b) For each transmission of the control message $\langle\{M^{(k)}\}_{k\in\mathbb{R}}\rangle$ there exists a real number m such that $M^{(k)} = active$ for $k > m$ and $M^{(k)} = passive$ for $k \leq m$.

Proof. (a) According to the initialization of the "sticky flag" termination detection algorithm, $S_i^{(k)}$ is initialized to *active* for those k for which P_i is initially k–*active*, and to *passive* otherwise. Thus, initially $S_i^{(k)}$ is *active* for $k > C_i$ and *passive* for $k \leq C_i$, hence (a) is satisfied with $s_i = C_i$.
The variable $S_i^{(k)}$ is assigned to only in line (6) of the algorithm and in action $\mathbf{R}_i$ of the underlying basic computation when P_i becomes k–active (see Section 3.1 and the modification to action $\bar{\mathbf{R}}_i$ in Section 2.2). Immediately after an execution of line (6), $S_i^{(k)} = active$ for $k > C_i$ and $S_i^{(k)} = passive$ for $k \leq C_i$, hence the value of C_i satisfies the requirement for s_i.
Now assume (a) is satisfied for some value s_i before action $\mathbf{R}_i$ and P_i becomes k–active for some k. This happens when C_i decreases (upon receipt of a basic

message). After this decrease the statement $S_i^{(k)} :=$ *active* must be executed for all $k > C_i$ (see action $\bar{\mathbf{R}}_i$ in Section 2.2 and the definition in Theorem 3.2). Thus, after this execution $S_i^{(k)} =$ *active* holds if and only if $k > s_i$ or $k > C_i$. (In the former case $S_i^{(k)} =$ *active* held already before the execution, in the latter case it became true as a result of the execution.) But $k > s_i \vee k > C_i$ is equivalent to $k > \min(s_i, C_i)$, hence (a) is now true if the value $\min(s_i, C_i)$ is substituted for the original value of s_i.

(b) When the control message is sent out by P_n, $M^{(k)} =$ *passive* for all k, hence (b) is satisfied for $m = \infty$.
The value of the $M^{(k)}$ is changed only in line (2) of the algorithm. We can assume that prior to the execution there exist values m and s_i such that $S_i^{(k)} =$ *active* iff $k > s_i$, and $M^{(k)} =$ *active* iff $k > m$. Hence after the execution $M^{(k)} =$ *active* iff $k > m$ or $k > s_i$. (In the former case $M^{(k)}$ was *active* already, in the latter case it became true as a result of the execution.) But $k > m \vee k > s_i$ is equivalent to $k > \min(m, s_i)$, so after the execution (b) is true if $\min(m, s_i)$ is substituted for the original value of m. □

The Sticky Flag GVT Approximation Algorithm. Theorem 4.1 is the key for obtaining a practicable and efficient *GVT* approximation algorithm. It shows that it is sufficient for each process to maintain the "boundary value" s_i as indicated in the theorem instead of a possibly infinite number of *passive*, *active* values. For a control message transmission it suffices to transmit the value m. The proof of the theorem describes directly how variables s_i and m change as the result of the combined action of the bundle given above. The resulting *GVT* approximation algorithm is rather simple. It is initiated by P_n, sending $\langle\infty\rangle$ to P_1. When receiving the circulating control message, P_i executes the following action $\mathbf{W}_i$ atomically.

```
W_i:  (1)  receive $\langle m \rangle$;
      (2)  $m := \min(m, s_i)$;
      (3)  if $i = n$ then $G := \max(G, m)$;
      (4)  if $i \neq n$ then send $\langle m \rangle$ to $P_{i+1}$
      (5)          else send $\langle \infty \rangle$ to $P_1$;
      (6)  $s_i := C_i$
```

The following two remarks can be made about the initialization and modification of the approximation G. (1) In order to satisfy the safety requirement, G must be initialized satisfying $G \leq k$ for all possible *GVT* values k. Thus G should be initialized to $\Leftrightarrow\infty$, or to zero if negative values do not appear. (2) It is a property of the algorithm that the statement $G := \max(G, m)$, which is called after each control round, is executed with non–decreasing values of m. As a result of this property, we can replace the statement $G := \max(G, m)$ by $G := m$. That G is still non–decreasing follows from Theorem 4.3 which is proved below.

This completes the derivation of the GVT approximation scheme. We continue this section with the superimposition that must be made on the three actions $\mathbf{I}_i$, $\mathbf{X}_i$, and $\mathbf{R}_i$ which model the behavior of a general distributed monotonic computation in order to approximate GVT.

The termination detection algorithm does not require any superimpositions on the actions $\bar{\mathbf{I}}_i$ and $\bar{\mathbf{X}}_i$, hence no modification of the corresponding actions $\mathbf{I}_i$ and $\mathbf{X}_i$ for GVT approximation is necessary. For the termination detection algorithm the statement $S_i := state_i$ was added to action $\bar{\mathbf{R}}_i$ (see Section 2.2). Thus for the case of GVT approximation the state indicators $S_i^{(k)}$ must be updated (if $t < C_i$ holds when the message with timestamp t is received) in action $\mathbf{R}_i$ in order to reflect the transition from k–passive to k–active for all $k \in (t, C_i]$. This is easily done for all relevant k in a single assignment to the variable s_i as indicated in the proof of Theorem 4.1.

```
$\mathbf{R}_i$:  **receive** $\langle t, \cdots \rangle$;
     $C_i := \min(C_i, t)$;
     $s_i := \min(s_i, C_i)$
```

Analogously to the original termination detection algorithm where the sticky state indicator is initialized to the current state, s_i should initially be set to the value of the local clock C_i.

4.2 Correctness of the Sticky Flag Algorithm

We will now prove the correctness of the algorithm presented above.

Lemma 4.2 *(a) $s_i \leq C_i$ is invariant in each process P_i.*
(b) Let GVT_0 denote the GVT at the start of the current control wave (i.e., at $\mathbf{W}_1$), and let P_i be the next process to be visited by the control wave. Then the following inequality holds invariantly:

$$GVT_0 \leq \min_{j=1}^{i-1} s_j$$

Proof. (a) Obvious from the actions and the initialization of the s_j.
(b) Initially the inequality holds vacuously as $min_{j=1}^{0} s_j$ is defined as ∞. Actions $\mathbf{I}_k$ and $\mathbf{X}_k$ do not change any variables involved. Action $\mathbf{R}_k$ may decrease s_k. Hence, if $k \leq i \Leftrightarrow 1$, $\min_{j=1}^{i-1} s_j$ may be decreased. However, if s_k is indeed changed, it is set to the current value of C_k, and hence to a value $\geq GVT$. As GVT is non–decreasing, $GVT \geq GVT_0$. Thus $\min_{j=1}^{i-1} s_j$ cannot be decreased to a value $< GVT_0$ by $\mathbf{R}_k$. Consider action $\mathbf{W}_i$, for $i \neq n$. As process P_i is visited here, "the next process to be visited" becomes P_{i+1} after execution of $\mathbf{W}_i$, and we have to prove that the inequality holds for i increased by 1. In $\mathbf{W}_i$, s_i is set to $C_i \geq GVT \geq GVT_0$, hence the inequality holds for the new value of i. Consider action

$\mathbf{W}_n$. Here GVT_0 is increased to GVT as a new control wave is started, and i is set to 1. For this new value of i the inequality holds vacuously. □

Theorem 4.3 *Let GVT_0 denote the GVT at the start of the current control wave (i.e., at $\mathbf{W}_1$), GVT_{-1} the GVT at the start of the previous control wave (defined as $\Leftrightarrow\infty$ if there is no previous one), GVT_{-2} the GVT at the start of the control wave before that (defined as $\Leftrightarrow\infty$ if there is none), and let P_i be the next process to be visited by the control wave. Then the following inequalities hold invariantly:*

$$GVT_{-2} \overset{(a)}{\leq} G \overset{(b)}{\leq} GVT_{-1} \overset{(c)}{\leq} \min(m, \min_{j=i}^{n} s_j) \overset{(d)}{\leq} \min(GVT_0, \min_{j=i}^{n} s_j) \overset{(e)}{\leq} \min_{j=1}^{n} s_j \overset{(f)}{\leq} GVT$$

Proof. Initially $GVT_{-2} = G = GVT_{-1} = \Leftrightarrow\infty$, $m = \infty$, $i = 1$, $s_k = C_k$ (for all k), and $GVT_0 = \min_{j=1}^{n} s_j = GVT$. Hence the inequalities hold. Notice that by Lemma 4.2(a) and the definition of GVT, inequality (f) is always true.

Consider action $\mathbf{I}_k$. There C_k is increased, and of the variables in the inequalities, only GVT might be increased as a result (recall that by definition $GVT = \min_{j=1}^{n} C_j$), thus all inequalities remain true.

Action $\mathbf{X}_k$ does not change any of the variables involved.

Consider action $\mathbf{R}_k$. Variable G is not changed, hence (a) and (b) remain true. Variable s_k is set to $\min(s_k, C_k)$. Consider the following two possibilities. First, s_k remains the same. Then all inequalities remain unchanged and true. Second, s_k is set to C_k, whereby it is decreased. As $C_k \geq GVT$ by definition, $s_k \geq GVT$ still holds. As GVT is monotonically increasing, we also have $s_k \geq GVT_0 \geq GVT_{-1}$. Hence (c) remains true and the value of $\min(GVT_0, \min_{j=i}^{n} s_j)$ is not changed. Because $\min(m, \min_{j=i}^{n} s_j)$ can only decrease, (d) continues to hold. Property $s_k \geq GVT_0$ also guarantees that (e) remains true.

Consider action $\mathbf{W}_i$ for $i \neq n$. Here process P_i is visited by the wave, and "the next process to be visited by the wave" becomes P_{i+1}. Thus we have to prove that the inequalities hold for a value of i that is increased by 1. Since line (3) does not apply, G is not changed and (a), (b) remain true. In line (2) m is assigned the value $\min(m, s_i)$. Hence the value of $\min(m, \min_{j=i}^{n} s_j)$ before the execution of line (2) is the same as $\min(m, \min_{j=i+1}^{n} s_j)$ after line (2). Note that line (6) does not affect the inequality $GVT_{-1} \leq \min(m, \min_{j=i+1}^{n} s_j)$. Hence (c) is kept invariant. Inequality (d) remains true because the value of the right-hand side can only increase when substituting i by $i+1$, whereas then the value of the left-hand side does not change as shown above. For (e) we observe that by Lemma 4.2(b) we have $GVT_0 \leq \min_{j=1}^{i-1} s_j$ and hence $\min(GVT_0, \min_{j=i+1}^{n} s_j) \leq \min_{j=1}^{n} s_j$. (Note that in line (6) s_i is set to a value $C_i \geq GVT \geq GVT_0$.) Hence inequality (e) does also hold if we set i to $i + 1$.

Consider action $\mathbf{W}_n$. Here process P_n is visited and P_1 becomes the next process to be visited. We denote the value of a variable v before the execution of $\mathbf{W}_n$ by v'. Because a new control wave is started by $\mathbf{W}_n$, GVT_{-2}, GVT_{-1}, and

GVT_0 attain the values of the old GVT'_{-1}, GVT'_0, and GVT'. As we had $GVT'_{-1} \leq \min(m', s'_n)$ before the action, the new value of $m = \min(m', s'_n)$ (after line (2)) and hence the new value of $G = m$ (line (3)) are $\geq GVT'_{-1} = GVT_{-2}$. Hence (a) is true. As before $\mathbf{W}_n$ $\min(m', s'_n) \leq \min(GVT'_0, s'_n) \leq GVT'_0$, we now have $G = m = \min(m', s'_n) \leq GVT'_0 = GVT_{-1}$. Hence (b) is true. From Lemma 4.2(b) we have $GVT'_0 \leq \min_{j=1}^{n-1} s_j$. As s_n is set to C_n and $C_n \geq GVT \geq GVT'_0$, now $GVT'_0 \leq \min_{j=1}^{n} s_j$. Hence $GVT_{-1} = GVT'_0 \leq \min_{j=1}^{n} s_j = \min(\infty, \min_{j=1}^{n} s_j)$ which proves that (c) holds for $i = 1$ and $m = \infty$. For the start of the new wave we set $i := 1$, $m := \infty$, $GVT_{-2} := GVT'_{-1}$, $GVT_{-1} := GVT'_0$, and $GVT_0 := GVT'$. As $\min_{j=1}^{n} s_j \leq \min_{j=1}^{n} C_j = GVT = GVT' = GVT_0$ (observe that GVT=GVT' because the clock values C_j do not change in action $\mathbf{W}_n$), we have $\min(m, \min_{j=1}^{n} s_j) = \min_{j=1}^{n} s_j = \min(GVT_0, \min_{j=1}^{n} s_j)$. Thus inequalities (d) and (e) do also hold. □

Corollary 4.4 *The presented algorithm is a correct GVT approximation algorithm.*

Proof. For the safety (property $\mathcal{G}1$, see Section 3.1) we note that $G \leq GVT$ follows directly from Theorem 4.3. To prove liveness (property $\mathcal{G}2$), consider the leftmost inequality $GVT_{-2} \leq G$ of Theorem 4.3. It implies that when GVT reaches some value, the approximation G will have reached this value after two rounds have been completed. From this and the monotonicity of G and GVT follows liveness. □

This correctness proof also applies to the "sticky flag" termination detection algorithm presented in Section 2.3 since termination detection is a particular instance of GVT approximation.

4.3 The DFG-Algorithm

Termination Detection. In [11] Dijkstra, Feijen, and van Gasteren presented a ring-based termination detection algorithm for distributed computations with instantaneously transmitted activation messages. Their algorithm is similar to the ring-based "sticky flag" algorithm developed in Section 2.3. The basic idea[4] of the DFG-algorithm is that a process P_i sets a flag $\hat{S}_i$ to *active* whenever it might reactivate a process that the wave has already visited. This is the case whenever it sends a message to some process P_j such that $j < i$. $\hat{S}_i$ is reset to *passive* when the circulating control message which collects the states and the flags of all processes is propagated from P_i to $P_{(i \ mod \ n)+1}$. If after a complete round the accumulated value of the control message is *passive*, then all processes were visited when they were *passive* and no process has been reactivated after the wave, hence termination can be concluded.

[4] We sketch the algorithm in a slightly different way than it was originally presented in [11].

The main difference with the "sticky flag" algorithm is that the DFG-algorithm uses *send-flags* $\hat{S}_i$, whereas our algorithm uses *receive-flags* S_i. It is easy to see that if $\hat{S}_i$ is *active*, then S_i would be *active* too. The converse, however, is not true; Figure 1 depicts a scenario in which, at the second wave, all S_i are *active* and all $\hat{S}_i$ are *passive*. In this example, the DFG-algorithm detects termination already at the end of the second wave whereas the "sticky flag" algorithm detects termination only at the end of the third wave.

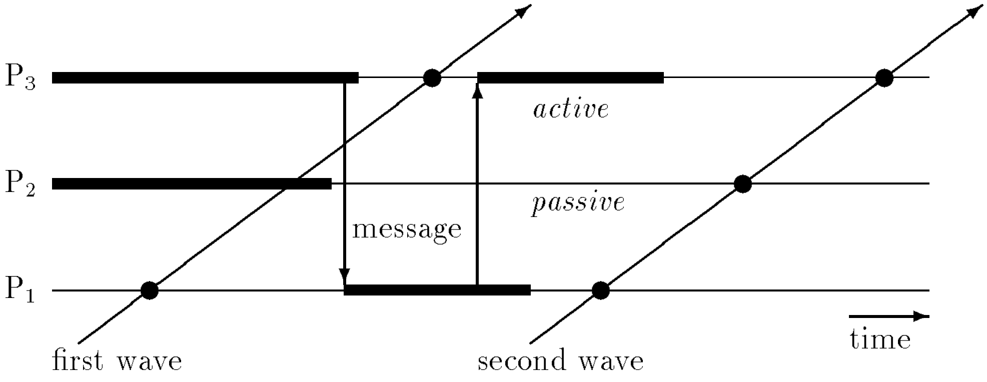

Figure 1: Termination detection based on control waves.

GVT Approximation. In the same way as the "sticky flag" algorithm, the DFG-algorithm can be transformed into a GVT approximation algorithm. The internal action $\mathbf{I}_i$ remains unchanged. The update of $\hat{s}_i$ now no longer takes place in the receive action $\hat{\mathbf{R}}_i$, but in the send action $\hat{\mathbf{X}}_i$.

```
$\hat{\mathbf{R}}_i$:  receive $\langle t, \cdots \rangle$;
      $C_i := \min(C_i, t)$

$\hat{\mathbf{X}}_i$:  choose $t \geq C_i$ ;
      send $\langle t, \cdots \rangle$ to $P_j$;
      if $j < i$ then $\hat{s}_i := \min(\hat{s}_i, t)$
```

According to the DFG-algorithm, the wave reflects both the value of the state and the flag $\hat{S}_i$, and as $\hat{s}_i \leq C_i$ does not necessarily hold for this algorithm, the state of a process (i.e., C_i) has to be considered in addition to $\hat{s}_i$ when accumulating the minimum in action $\hat{\mathbf{W}}_i$:

```
$\hat{\mathbf{W}}_i$:  (1)  receive $\langle \hat{m} \rangle$;
      (2)  $\hat{m} := \min(\hat{m}, \hat{s}_i, C_i)$;
      (3)  if $i = n$ then $\hat{G} := \max(\hat{G}, \hat{m})$;
```

```
(4)                 send $\langle\infty\rangle$ to $P_1$
(5)             else  send $\langle\hat{m}\rangle$ to $P_{i+1}$;
(6)   $\hat{s}_i := \infty$
```

Setting $\hat{S}_i$ to *passive* in the DFG-algorithm transforms into $\hat{s}_i := \infty$ in the last line of action $\hat{\mathbf{W}}_i$.

Correctness. It is interesting to note that a direct transformation of the invariant given in [11] for the correctness of the termination detection algorithm leads to an invariant implying the correctness of the GVT approximation version given above.

Theorem 4.5 *Let GVT_0 denote the GVT at the start of the current control wave (i.e., at $\mathbf{W}_1$), GVT_{-1} the GVT at the start of the previous control wave (defined as $\Leftrightarrow\infty$ if there is none), GVT_{-2} the GVT at the start of the control wave before that (defined as $\Leftrightarrow\infty$ if there is none), and let P_i be the next process to be visited by the control wave. Then the following inequalities hold invariantly:*

$$GVT_{-2} \leq \hat{G} \leq GVT_0 \leq \min_{j=1}^{i-1} \hat{s}_j$$

$$GVT_{-1} \leq \min(\hat{m}, \min_{j=i}^{n} \hat{s}_j) \leq \min_{j=1}^{i-1} C_j$$

Proof. Initially, $i = 1$, $GVT_{-2} = GVT_{-1} = \hat{G} = \Leftrightarrow\infty$, $GVT_0 = GVT$, $\hat{m} = \hat{s}_j = \infty$ for all j, while $\min_{j=1}^{0}$ is defined as ∞ for all arguments.

Action $\hat{\mathbf{I}}_k = \mathbf{I}_k$ increases C_k which leaves the inequalities true.

As message transmission is instantaneous, we only consider send actions together with the corresponding receive action. Hence, consider action $\hat{\mathbf{X}}_k\hat{\mathbf{R}}_h$ where a message is sent from P_k to P_h. In $\hat{\mathbf{R}}_h$ C_h might be decreased which could have an effect on $\min_{j=1}^{i-1} C_j$ if $h < i$. We distinguish two cases.

Case (1): $h < k$. Then $\hat{s}_k$ is set to $\min(\hat{s}_k, t)$, but if it is changed, it cannot be decreased beyond GVT_0 as $t \geq C_k$. If $\min_{j=1}^{i-1} C_j$ is actually decreased because C_h is set to t, then we have that $h < i \leq k$ and (because of the last line in action $\hat{\mathbf{X}}_k$) $\min(\hat{m}, \min_{j=i}^{n} \hat{s}_j) \leq t$. (Note that $\min_{j=1}^{i-1} C_j$ is not decreased if $k < i$ because then the timestamp t sent by $\hat{\mathbf{X}}_k$ and received by $\hat{\mathbf{R}}_h$ is $\geq C_k$).

Case (2): $h > k$. Only C_h might be decreased, but not $\min_{j=1}^{i-1} C_j$ for the same reason as mentioned above ($i > h > k$).

Hence the inequalities continue to hold.

Consider action $\hat{\mathbf{W}}_i$ for $i \neq n$. Here process P_i is visited by the control wave and "the next process to be visited" becomes P_{i+1}, hence we have to prove that the inequalities hold for i increased by 1. Since $\hat{G}$ is not changed, the first and the second inequality continue to hold. In line (2) $\hat{m}$ may be decreased, but $\hat{s}_i$ was already included in $\min(\hat{m}, \min_{j=i}^{n} \hat{s}_j)$ before the action, and C_i is added to both $\hat{m}$ (line (2)) and $\min_{j=1}^{i-1} C_j$ when i is increased by 1 in line (5). Hence the fifth inequality continues to hold. If $\min(\hat{m}, \min_{j=i}^{n} \hat{s}_j)$ is actually decreased that can only be due to the inclusion of C_i in the minimum, hence it cannot be decreased beyond GVT_0 and the fourth inequality remains true. As $\hat{s}_i$ is set to ∞, the third inequality also holds for i increased by 1.

Consider action $\hat{\mathbf{W}}_n$. Here process P_n is visited and a new control wave is started. Hence we have to prove that the inequalities hold for $i = 1$ and the new values for GVT_{-2}, GVT_{-1}, GVT_0, and $\hat{G}$. The value of a variable v before $\hat{\mathbf{W}}_n$ will be denoted by v'. From the second invariant we have $GVT'_{-1} \leq \min(\hat{m'}, \hat{s'}_n) \leq \min_{j=1}^{n-1} C_j$ before line (2). Because $C_n \geq GVT'_{-1}$, this yields $GVT'_{-1} \leq \hat{m} = \min(\hat{m'}, \hat{s'}_n, C_n) \leq \min_{j=1}^{n} C_j = GVT = GVT'$ after line (2). As GVT_{-2}, GVT_{-1}, GVT_0, and $\hat{G}$ attain the values of GVT'_{-1}, GVT'_0, GVT', and $\hat{m}$, now the first invariant $GVT_{-2} \leq \hat{G} \leq GVT_0 \leq \min_{j=1}^{0} \hat{s}_j$ holds. From the third inequality we have $GVT'_0 \leq \min_{j=1}^{n-1} \hat{s'}_j$. As $\hat{s}_n$ and $\hat{m}$ are set to ∞ and $GVT_{-1} = GVT'_0$, this yields $GVT_{-1} \leq \min(\hat{m}, \min_{j=1}^{n} \hat{s}_j) \leq \min_{j=1}^{0} C_j = \infty$. Hence action $\hat{\mathbf{W}}_n$ preserves all inequalities. □

The safety and liveness now follow from $GVT_{-2} \leq \hat{G} \leq GVT_0 \leq GVT$ for the same reasons as for the "sticky flag" algorithm.

Corollary 4.6 *The presented algorithm is a correct GVT approximation algorithm.*

For the same reason why the DFG-algorithm might detect termination earlier than the algorithm of Section 2.3, this variant might yield a better approximation of the current GVT value ($\hat{G} \leq GVT_0$) than the algorithm of Section 4.1 ($G \leq GVT_{-1}$). Unfortunately, however, it does not generalize as easily to the asynchronous case as the "sticky flag" based scheme. We come back to this problem in the next section.

5 Discussion

In this section directions to extend the results of this paper are discussed, as well as the relation with some other work.

5.1 Extensions

Asynchronous Communication. The GVT approximation algorithms described so far require that activation messages be transmitted instantaneously. However, the algorithms are also applicable to synchronous communication and so-called causally ordered communication [2, 9], and they can also be adapted to the asynchronous case where messages may be in transit. In the asynchronous case, GVT is the minimum of all clocks C_i and of all timestamps of messages which are in transit. Hence, if all processes are *t–passive* and no t–activation messages are in transit then $GVT \geq t$. Whether t–activation messages are in transit can be checked using acknowledgements: Define a process P_i to be *t–engaged* if it is *t–active* or the receipt of a t–activation message sent by P_i is not yet acknowledged. Then $GVT \geq t$ holds if no process is *t–engaged.* Let $UNACK_i$ denote P_i's multiset of timestamps of unacknowledged messages. Then P_i is *x–engaged* for all $x > \min(\{C_i\} \cup UNACK_i)$, and not *x–engaged* for all $x \leq \min(\{C_i\} \cup UNACK_i)$.

For the "sticky flag" algorithm, approximation of GVT in the asynchronous case can be done in the same way as before, basically by substituting "*t–engaged*" for "*t–active*". Observe that a process which is not *t–engaged* can only become *t–engaged* if at that moment there exists another *t–engaged* process. Therefore, rule $\mathcal{R}2'$ (Section 2.4) is observed and hence the "sticky flag" algorithm can be applied to check whether $GVT \geq t$. For that purpose, only the last line in action $\mathbf{W}_i$ has to be changed into

$$s_i := \min(\{C_i\} \cup UNACK_i)$$

in order to reflect the new interpretation of the local state.

The DFG-algorithm has already been adapted so as to detect termination of computations using asynchronous communication [5]. In those variants, however, the control wave is "lazy" in the sense that a process does not propagate the control message as long as it is active or engaged. As the process is engaged all the time between the sending of a message and the receipt of the acknowledgement, it obviously does not matter at what moment the state indicator is assigned. A lazy execution of the control wave, however, makes the algorithm less suitable for our transformation.

The reason why a non–lazy variant of the DFG-algorithm does not generalize as easily to the asynchronous case as the "sticky flag" scheme is that in the DFG-algorithm the *sender* is responsible for setting the state indicator. It is not aware of the moment at which the message is received. Thus is is not obvious at what moment (between the sending of the message and the receipt of the acknowledgement) the state indicator $\hat{s}_i$ must be set if instantaneous message transmission is to be simulated. However, a possibility to cope with asynchronous communication in non–lazy versions of the DFG-algorithm is to reset the state indicator $\hat{s}_i$ to the current value of $\min(UNACK_i)$ (rather than ∞) after each visit of the control wave. In this way, the state indicator is logically assigned all times between the sending and the receipt of the acknowledgement, and thus in particular at the moment when the message is received. This essentially yields a GVT approximation scheme which was presented (without proof) by Bellenot in [3].

Another approach to adapt GVT approximation algorithms to the asynchronous case was used by Schoone and Tel in [29]. Instead of sending acknowledgements for messages, the timestamps of messages to be acknowledged are accumulated in multisets $ACK_i[j]$ (a process needs an entry for each originator). At a visit of the control message, these multisets are then transferred to the control message, thus sending all acknowledgements at once. Likewise, the multiset $UNACK_i$ is transferred to the control message upon a visit, whereby acknowledged and unacknowledged messages can cancel each other. For more details we refer to [29].

Lazy Algorithms. In this paper we have applied our transformation to two particular *non–lazy* algorithms where the control message is propagated independently of the state of the process. Most termination detection algorithms known to date can be made non–lazy by a simple modification, but there are inherently lazy algorithms with favorable properties, such as the algorithm by Dijkstra and Scholten [12]. Although the results of Section 3.3 are in principle applicable to lazy algorithms, the reduction to a finite algorithm is not so easy in this case. The application of the transformation to the algorithm of Dijkstra and Scholten is currently under investigation.

Distributed Infimum Approximation. The notion of a monotonic distributed computation can be generalized by replacing the domain $\mathbb{R}$ of the clocks and timestamps by an arbitrary partially ordered set. The resulting problem, of which termination detection and GVT approximation are instances, is called *distributed infimum approximation.* This problem was defined by Tel [30] and algorithms to approximate a distributed infimum were given in [30, 32].

It was already noted in [29] that a termination detection algorithm can be transformed to yield an algorithm for distributed infimum approximation. There, however, only the transformation for a particular algorithm (the so-called vector counter algorithm [20]) is shown, and that transformation yields a rather complicated algorithm for the general case of arbitrary posets and asynchronous communication. In the current paper we have demonstrated that the transformation is general enough so as to be applicable to *any* termination detection algorithm, and have exemplified it with a transformation yielding a simple, practical algorithm.

The generalization of our current work to distributed infima is straightforward as far as non–lazy termination detection algorithms are considered. The construction of a distributed infimum approximation algorithm from a lazy termination detection algorithm is left as a subject for further research.

5.2 Further Remarks

Related Work. A large number of termination detection algorithms have been published in recent years, and many of those published before 1987 are listed in the bibliography in [20]. As we showed in Section 4.3, our "sticky flag" algorithm (which was also derived in [33] when applying to Ben-Ari's garbage collection algorithm [4] a scheme that systematically transforms garbage collection algorithms into distributed termination detection algorithms) is similar to the termination detection algorithm by Dijkstra, Feijen, and van Gasteren [11]. Because of its simplicity (messages need not be considered; a single flag is used which is only set when a process actually becomes active) our algorithm compares favorably with this and other termination detection algorithms based on synchronous communication.

Some of the ideas that are developed in this paper were already used, often implicitly, in earlier papers. The idea of using a bundle of termination detection algorithms to approximate GVT did already appear in a distributed garbage collection algorithm by Hughes [15]. This algorithm, however, which is based on Rana's termination detection scheme [24], requires a global clock. The idea has also been sketched by Chandy and Sherman [7] although their resulting algorithm is not used for GVT approximation. Connections between termination detection and GVT approximation were made by Jefferson [16] and Tel [30, 32] and, as mentioned above, by Schoone and Tel [29].

The idea of using acknowledgements to catch the timestamps of in-transit messages in GVT approximation schemes was already used by Samadi et al. [27]. Other solutions to the GVT approximation problem were given by Bauer et al. [1], Conception and Kelly [10], Lin and Lazowska [18], Preiss [23], and, as already mentioned above, by Bellenot [3] and Baldwin et al. [2]. Most of these solutions, however, are either rather involved or not proved to be correct. Because of the importance of a fast and efficient GVT approximation for distributed simulation, hardware solutions have also been proposed ([13, 19, 26]).

Since GVT is a monotonic function of the global state, it is also possible to use distributed snapshot algorithms (as given by Chandy and Lamport [6] or Lai and Yang [17]) in order to approximate GVT. In [22] a snapshot based solution for asynchronous communications is proposed which does not rely on acknowledgements. The results of Lai and Yang [17] indicate that for the detection of termination it is not necessary to compute a so–called *consistent* snapshot. As is shown in [22], inconsistent snapshots are also sufficient for GVT approximation.

Conclusions. In this paper algorithms for termination detection and GVT approximation were studied; their correctness was proved using an assertional verification technique. It was first shown that distributed termination detection is, although non–trivial, not a complicated problem; the "sticky flag" paradigm was introduced, which allows to design termination detection algorithms that are easily understood from an intuitive point of view. We also showed how algorithms for instantaneous (or synchronous) message transmission can be transformed into algorithms for asynchronous communications.

Computing a lower bound on the *GVT* of a distributed monotonic computation is a generalization of the distributed termination detection problem. In principle, any termination detection algorithm can be used to check whether some "guessed" value k is a lower bound on the *GVT* by determining whether all processes are k–passive (or no process is k–engaged). It is then possible to run detection algorithms for many different values of k in parallel and to take the best approximation.

We demonstrated that the synchronized execution of termination detection algorithms based on the "sticky flag" paradigm or a similar principle can be simulated in a practicable way yielding efficient *GVT* approximation algorithms. The principle of our *GVT* approximation scheme is rather simple: Each process remembers the smallest value its clock has assumed since the last visit of the wave. The wave collects the minimum of all those values which is then taken as a new approximation of the current *GVT*. This simple and provably correct scheme compares favorably with other known *GVT* approximation algorithms. It is left for further investigation whether termination detection schemes based on other principles, such as the diffusing computation paradigm [12] or the credit recovery paradigm [21], do also yield interesting *GVT* approximation algorithms.

References

[1] Bauer, H., Sporrer, C., and Krodel, T. H. On distributed logic simulation using time warp. In *Proc. VLSI International Conference (IFIP), Edinburgh* (1991).

[2] Baldwin, R., Chung, M. J., and Chung, Y. Overlapping window algorithm for computing GVT in time warp. In *11th International Conference on Distributed Computing Systems* (1991).

[3] Bellenot, S. Global virtual time algorithms. In *Proc. of the SCS Multiconference on Distributed Simulation* (1990), pp. 122–127.

[4] Ben-Ari, M. Algorithms for on–the–fly garbage collection. *ACM Trans. Program. Lang. Syst. 6* (1984), 333–344.

[5] Blanc, P. *Détection de Propriétés de Repos Globales dans des Systèmes Répartis avec Déséquencement de Messages; Application au Problème de Terminaison.* PhD thesis, Université Paris 6, Paris, 1990.

[6] Chandy, K. M., and Lamport, L. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst. 3* (1985), 63–75.

[7] Chandy, K. M., and Sherman, R. Space–time and simulation. In *Proc. of the SCS Multiconference on Distributed Simulation* (1989), pp. 53–57.

[8] Chang, E. J.-H. Echo algorithms: Depth parallel operations on general graphs. *IEEE Trans. Softw. Eng. SE–8* (1982), 391–401.

[9] Charron-Bost, B., Tel, G., and Mattern, F. Synchronous and asynchronous communication in distributed computations. Tech. rep., Université Paris 7, Paris, 1991.

[10] Concepcion, A. I., and Kelly, S. G. Computing global virtual time using the multilevel token passing algorithm. In *Proc. of the SCS Multiconference on Distributed Simulation* (1991), pp. 63–68.

[11] Dijkstra, E. W., Feijen, W. H. J., and van Gasteren, A. J. M. Derivation of a termination detection algorithm for distributed computations. *Inf. Process. Lett. 16* (1983), 217–219.

[12] Dijkstra, E. W., and Scholten, C. S. Termination detection for diffusing computations. *Inf. Process. Lett. 11* (1980), 1–4.

[13] Filloque, J. M., Gautrin, E., and Pottier, B. Efficient global computations on a processor network with programmable logic. In *Proc. Parallel Architectures and Languages Europe (PARLE)* (1991), E. H. L. Aarts, J. van Leeuwen, and M. Rem, Eds., vol. 505 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 69–82.

[14] Fujimoto, R. M. Parallel discrete event simulation. *Commun. ACM 33*, 10 (1990), 30–53.

[15] Hughes, J. A distributed garbage collection algorithm. In *Functional Programming Language and Computer Architecture* (1985), J. P. Jouannoud, Ed., vol. 201 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 256–272.

[16] Jefferson, D. Virtual time. *ACM Trans. Program. Lang. Syst. 7*, 3 (1985), 404–425.

[17] Lai, T. H., and Yang, T. H. On distributed snapshots. *Inf. Process. Lett. 25* (1987), 153–158.

[18] Lin, Y. B., and Lazowska, D. Determining the global virtual time in a distributed simulation. Tech. Rep. 90–01–02, Dept. of Comp. Sc., Univ. of Washington, Seattle, 1990.

[19] Livny, M., and Manber, U. Distributed computation via active messages. *IEEE Trans. on Computers C-34*, 12, (1985), 1185–1190.

[20] Mattern, F. Algorithms for distributed termination detection. *Distributed Computing 2* (1987), 161–175.

[21] Mattern, F. Global quiescence detection based on credit distribution and recovery. *Inf. Process. Lett. 30* (1989), 195–200.

[22] Mattern, F. Efficient distributed snapshots and global virtual time algorithms. Tech. Rep. SFB124–24/90, University of Kaiserslautern, Kaiserslautern, 1990.

[23] Preiss, B. R. The Yaddes distributed discrete event simulation specification language and execution environments. In *Proc. of the SCS Multiconference on Distributed Simulation* (1989), pp. 139–144.

[24] Rana, S. P. A distributed solution of the distributed termination problem. *Inf. Process. Lett. 17* (1983), 43–46.

[25] Raynal, M., and Hélary, J.-M. *Control and Synchronization of Distributed Systems and Programs.* Wiley, 1990.

[26] Reynolds, P. F. An efficient framework for parallel simulation. In *Proc. of the SCS Multiconference on Advances in Parallel and Distributed Simulation* (1991), pp. 167–174.

[27] Samadi, B., Muntz, R. R., and Parker, D. S. A distributed algorithm to detect a global state of a distributed simulation system. In *Proceedings IFIP Conference on Distributed Processing* (Amsterdam, 1987), North–Holland.

[28] Schoone, A. A. *Assertional Verification in Distributed Computing.* PhD thesis, University of Utrecht, May 1991.

[29] Schoone, A. A., and Tel, G. Transformation of a termination detection algorithm and its assertional correctness proof. Tech. Rep. RUU–CS–88–40, University of Utrecht, Utrecht, 1988. Also [28, Sec. 5.2].

[30] Tel, G. Distributed infimum approximation. In *Fundamentals of Computation Theory* (Kazan, 1987), L. Budach et al., Eds., vol. 278 of *Lecture Notes in Computer Science*, Springer–Verlag, pp. 440–447. Also [32, Sec. 4.1].

[31] Tel, G. Total algorithms. *Algorithms Review 1* (1990), 13–42. Also [32, Sec. 4.2].

[32] Tel, G. *Topics in Distributed Algorithms*, vol. 1 of *Cambridge International Series on Parallel Computation.* Cambridge University Press, Cambridge, U.K., 1991.

[33] Tel, G., and Mattern, F. The derivation of termination detection algorithms from garbage collection schemes. Tech. Rep. RUU–CS–90–24, University of Utrecht, Utrecht, 1990. (Submitted to: ACM Trans. on Prog. Lang. and Syst.).

Contents

1 Introduction **2**

2 Termination Detection **3**

2.1 The Problem 3

2.2 A Simple Termination Detection Principle 4

2.3 A Termination Detection Algorithm 6

2.4 Variants 7

3 Global Virtual Time and its Approximation **8**

3.1 The Global Virtual Time Problem 8

3.2 The Relation between GVT and Termination 10

3.3 GVT Approximation with Termination Detection 12

4 Two Simple GVT Approximation Algorithms **15**

4.1 Transformation of the Sticky Flag Algorithm 15

4.2 Correctness of the Sticky Flag Algorithm 19

4.3 The DFG-Algorithm 21

5 Discussion **24**

5.1 Extensions 24

5.2 Further Remarks 27

References **28**