Knowledge-based Specification: Investigating Distributed Mutual Exclusion (Extended Abstract)

Umberto Bonollo
Department of Computer Science
and Software Engineering
The University of Melbourne
Melbourne 3010 Australia
umberto@cs.mu.oz.au

Ron van der Meyden
School of Computer Science
and Engineering
University of New South Wales
Sydney 2052 Australia
meyden@cse.unsw.edu.au

Liz Sonenberg
Department of Information Systems
The University of Melbourne
Melbourne 3010 Australia
l.sonenberg@dis.unimelb.edu.au

June 6, 2001

Abstract

We adopt the knowledge-based approach to distributed systems' analysis. Close examination of a class of practical algorithms for distributed mutual exclusion leads to a knowledge-based specification of the class. The specification is obtained by focusing on the knowledge content of what needs to be communicated at each stage of the protocol. Correctness results are outlined, including our main result on liveness. We discuss how this work sheds light on the subtle nature of the standardly presented algorithms and so provides further evidence of the value of the knowledge-based approach.

1 Introduction

Fagin, Halpern, Moses and Vardi [1995] laid the groundwork for the *knowledge-theoretic approach* to distributed systems: a high-level approach to distributed system analysis that enables a more abstract, and more insightful, perspective on the design of distributed protocols. The approach focuses on *knowledge* and, in particular, on how *communication* changes the states of knowledge of each of the processes. Advantages include: enabling a deeper understanding of why a distributed algorithm is correct; facilitation of the redesign of distributed algorithms for varying environments; and facilitation of the implementation and optimisation of distributed algorithms [Vardi, 1996]. Several distributed system problems have been successfully studied using this approach (e.g., [Halpern and Zuck, 1992]). However, the number of examples remains small and it is crucial for the continued development of the knowledge-theoretic approach that it be applied to the specification and design of additional pragmatic distributed system problems [Moses, 1997]. In this paper, we develop an additional application of the knowledge-based approach.

We provide a knowledge-based specification for an important subclass of algorithms for the *distributed mutual exclusion* (DME) problem. The specification is of a class of algorithms introduced earlier [Bonollo and Sonenberg, 1996], and related to the work on information structure DME algorithms by Sanders [1987]. The key to this work is to carefully identify the interrelationships between various properties of the processes (e.g. Sections 2.3, 3.2 and 3.3 below), the various assumptions as to 'who goes next' (Sections 3.4 and 4.2) and the memory requirements (Section 4.3).

The existing presentation [Bonollo and Sonenberg, 1996] of the class of algorithms we study is standard, in the sense that it is cast in terms of processes with a set of states described by a concrete set of typed variables and an imperative program describing the behaviour of the processes. Our contribution in this paper is to lift this description to a much higher level of abstraction, where we focus on the flow of information in the system. In place of messages with concretely given syntax, we focus on the semantic content of what needs to be communicated at each state of the protocol, using a "notify" primitive similar to that introduced by Moses and Kislev [1993]. We also express the conditions under which certain actions are taken (specifically, entering the critical section) in terms of the knowledge that a process must have to safely execute that action. This yields an abstract knowledge-based description of the protocol for which we may then seek concrete implementations. Developing such implementations remains to be undertaken, but our hope is that the effort of abstraction will be repaid by allowing the discovery of implementations that provide some significant optimizations of the concrete protocol with which we started (like identification of situations in which some messages need not be sent, or in which action can be taken based on receipt of a smaller set of messages than expected.) Whether or not such benefits eventuate, the abstract specification has already given us better insight into the protocol.

Below we review relevant work on DME algorithms, then in the next section, we present the definitions for an epistemic temporal logic for describing properties of distributed systems. This is followed by a knowledge-theoretic specification of DME algorithms with timestamps, which includes a proof of mutual exclusion. We then present a knowledge-theoretic specification of the topology and protocol for DF-ISDME algorithms. The main result of the paper, a liveness proof that any process trying to enter the CS will eventually do so, is outlined. We then discuss how the knowledge-theoretic approach yields a deeper understanding for this pragmatic class of algorithms, and point to future work.

1.1 Distributed Mutual Exclusion

To achieve distributed mutual exclusion, a set of distributed processes must coordinate their actions such that at any time at most one process has access to a shared resource: for example, a network printer, or a tuple in a distributed database. A process may only be within its *critical section (CS)* whenever it has gained exclusive access to the shared resource. Typically, processes can only communicate with each other by message passing - there is no shared memory.

Several well known distributed mutual exclusion algorithms have been published and a taxonomy of algorithms appears in the survey by Singhal [1993]. Generally, these algorithms can be classed as either *token-based* or *information structure (ISDME)* algorithms. In token-based algorithms, a unique token is passed among the processes. Possession of the token guarantees mutual exclusion. The token-based algorithm of Suzuki and Kasami is a well known example. Information structure algorithms are instances of a general algorithm (GEN-ISDME) proposed by Sanders [1987] as a means of classifying several non-token-based algorithms. Among the advantages of information structure algorithms is their potential for local handling of process failure and process recovery while, by comparison, token-based algorithms require global cooperation for handling process failure which can be quite complex [Sanders, 1987].

In an ISDME algorithm, each process has a pair of sets called the *request set* and the *inform set*, which form the *local* information structure. The collection of these pairs defines the *global* information structure. For the GEN-ISDME algorithm, each process requests permission to enter its CS by sending *Request* messages to the processes which are members of its request set followed by waiting for *Grant* messages to be received from them. Upon exiting its CS, the process will then send *Release* messages to the processes which are members of its inform set. Mutual exclusion is only guaranteed when the request sets and the inform sets satisfy a certain condition on the global information structure. It turns out that the request sets and inform sets can describe the topology of several well known and practical mutual exclusion algorithms [Sanders, 1987].

Unfortunately, due to arbitrary communication delays, the GEN-ISDME algorithm can deadlock under certain information structures [Sanders, 1987] and so is impractical. Sanders proposed a deadlock-detecting algorithm (DDGEN-ISDME) which detects and avoids potential deadlock situations: it uses a deadlock recovery scheme requiring the exchange of Inquire, Yield, and Fail messages. The deadlock recovery scheme is quite complex. Our earlier work identified a class of Deadlock-free ISDME algorithms (DF-ISDME) which do not require any deadlock recovery scheme. These algorithms operate on a more restricted class of information structures than the class introduced by Sanders for GEN-ISDME, but their performance (via conventional measures such as message performance and synchronisation delays) is acceptably good [Bonollo and Sonenberg, 1996]. Below we present a formal specification of the class of systems which describe DF-ISDME algorithms.

The contribution of the present paper is to provide a knowledge-based specification of this practical class of algorithms, and in doing so both to shed light on the subtle nature of the algorithms, and to provide further evidence of the value of the knowledge-based approach to distributed systems' analysis. For example, the impact of the most complex axiom in our specification, the Dictate axiom, allows a process holding a lower priority timestamp to sometimes "jump ahead" of a higher priority process when messages from the higher priority process are delayed in transit—but this behaviour isn't mentioned at all in the standard algorithm proofs in the conventional literature. Instead, the standard algorithm proofs tend to imply (by omission) that all requesting processes will enter the CS in timestamp order.

2 Definitions

2.1 Knowledge and Temporal Properties

We use a first-order modal logic of knowledge and time along with the usual S5 axioms [Fagin *et al.*, 1995]. Our *until* $\mathcal{U}$ and *waiting-for* $\hat{\mathcal{W}}$ temporal operators are as in [Manna and Pnueli, 1992]. We use i and j to denote rigid variables which are quantified over the set of processes.

2.2 Edge Formulas

Often, we are interested in time points where a condition has just become true, or alternately we are interested in time points where a condition has just become false. The following definitions represent such "edge formulas:" The proposition that "ϕ has just become true" is represented as *enter*(ϕ) and is defined as follows: $\mathit{enter}(\phi) \overset{\text{defn}}{\equiv} \ominus\neg\phi\wedge\phi$. The proposition that

"ϕ has just become false" is represented as $exit(\phi)$ and is defined as follows: $exit(\phi) \overset{\text{defn}}{\equiv} \ominus\phi \wedge \neg\phi$. We define the proposition *first* such that it is true only at the initial point of an execution (or *run*).

2.3 Perfect Recall

Intuitively, *perfect recall* means that the local state of a process encodes everything that has happened—from the point of view of the process—thus far in the run. Thus the state of the process at time $m + 1$ contains at least as much information as its state at time m [Fagin *et al.*, 1995].

For a process i, the following properties are valid in systems with perfect recall [Halpern *et al.*, 1997]:

$$\forall i\,(K_i\,(\phi)\,\mathcal{S}K_i\,(\psi) \Rightarrow K_i\,(K_i\,(\phi)\,\mathcal{S}K_i\,(\psi))) \qquad \text{(Perfect Recall)}$$

$$\forall i\,(K_i\,(\Box\phi) \Rightarrow \Box K_i\,(\phi)) \qquad \text{(KT1)}$$

2.4 Notify

Moses and Kislev [1993] introduced the concept of the *notify* formula. This formula represents a way of abstracting the details of how two processes communicate by focusing on the transition of knowledge which must occur between them. Here:

$$notify_{ij}(\phi) \overset{\text{defn}}{\equiv} K_i\,(\phi) \wedge \Diamond K_j\,(\phi) \qquad \text{(Notify)}$$

3 Distributed Mutual Exclusion

This section presents a knowledge-theoretic specification (Properties 1 to 13) for the class of systems which describes DME algorithms using timestamps (DMET). From this specification, we provide a proof that mutual exclusion follows.

3.1 Timestamps

Each process of a DMET system which is trying to enter its critical section (CS) will make use of a timestamp. A *timestamp* (i, x) is an ordered pair consisting of a process identifier $i \in \mathrm{A} = \{1, \ldots, n\}$ and a natural number $x \in \mathbf{N}$. Timestamps are elements of a well-ordered domain such that $(i, x) < (j, y) \overset{\text{defn}}{\equiv} (x < y) \vee ((x = y) \wedge (i < j))$. We use t and t' to denote rigid variables which are quantified over the set of timestamps.

3.2 Process States

Processes in a DMET system are at any time in at most one of the following three local states: These states are called *Noncritical (N)*, *Trying (T)*, or *Critical Section (CS)*. A process i always uses a timestamp t whenever it is in its Trying state or in its CS state. We define several primitive *local propositions* to represent these concepts—intuitively, a proposition p is local to a process if its truth depends entirely on the local state of that process. Hence we write, *p is local to i* if and only if $\forall i(K_i\,(p) \vee K_i\,(\neg p))$ is valid. The local proposition that "process i is currently in a Noncritical state" is represented as N_i. The local proposition that "process i is currently in a Trying state with a timestamp t" is represented as $T_i(t)$. The local proposition that "process i is currently in the CS with a timestamp t" is represented as $CS_i(t)$. The local proposition that "process i is currently in a trying state [with some timestamp]" is represented as T_i and is defined as follows: $T_i \overset{\text{defn}}{\equiv} \exists t(T_i(t))$. The local proposition that "process i is currently in the CS [with some timestamp]" is represented as CS_i and is defined as follows: $CS_i \overset{\text{defn}}{\equiv} \exists t(CS_i(t))$. These propositions are related by the following general properties.

Property 1 For a process j, at any time, at most one of N_j, T_j, or CS_j holds.

$$\forall j((N_j \wedge \neg T_j \wedge \neg CS_j) \vee (\neg N_j \wedge T_j \wedge \neg CS_j) \vee (\neg N_j \wedge \neg T_j \wedge CS_j)) \tag{NTC}$$

Initially, every process starts in the Noncritical state as shown by the following property.

Property 2 If this is the first state of a run then process j is in the Noncritical state.

$$\forall j(\mathit{first} \Rightarrow N_j) \tag{Initial}$$

The states of a process happen in a cycle that repeats in the sequence N, T and CS. This cycle is captured by the following state transition axioms.

Property 3 If a process j is in the Noncritical state then it remains in the Noncritical state indefinitely or until it enters the Trying state.

$$\forall j(N_j \Rightarrow N_j \hat{\mathcal{W}} T_j) \tag{NT}$$

Property 4 If a process j is in the Trying state with a timestamp t' then it remains in the Trying state with timestamp t' indefinitely or until it enters the CS with timestamp t'.

$$\forall j, t'(T_j(t') \Rightarrow T_j(t') \hat{\mathcal{W}} CS_j(t')) \tag{TC}$$

Property 5 If a process j is in the CS with a timestamp t' then it remains in the CS with timestamp t' indefinitely or until it enters the Noncritical state.

$$\forall j, t'(CS_j(t') \Rightarrow CS_j(t') \hat{\mathcal{W}} N_j) \tag{CN}$$

The time spent by a process j in the CS is bounded.

Property 6 If a process j is in the CS with a timestamp t' then eventually it will reach a state where it is in the CS with timestamp t' and in the next time point it will be in the Noncritical state.

$$\forall j, t'(CS_j(t') \Rightarrow \Diamond(CS_j(t') \wedge \bigcirc N_j)) \tag{Bounded-CS}$$

3.3 Timestamp Properties

Each process generates its own unique set of timestamps as described by the following property.

Property 7 It's always the case that distinct processes i and j have distinct timestamps.

$$\forall i, t, j, t'(i \neq j \Rightarrow t \neq t') \tag{UT}$$

The following two properties cover the fact that, at any time, a process i is using only a single timestamp within a process state.

Property 8 At each point in time, each process is in the Trying state with at most one timestamp (if at all).

$$\forall i, t, t' (T_i(t) \wedge T_i(t') \Rightarrow t = t') \tag{EQTT}$$

Property 9 At each point in time, each process is in the CS with at most one timestamp (if at all).

$$\forall i, t, t' (CS_i(t) \wedge CS_i(t') \Rightarrow t = t') \tag{EQTCS}$$

Monotonicity of timestamps is a key property which is expressed as follows:

Property 10 Given (NTC), whenever a process revisits the Trying state after having visited the CS, it will use a timestamp greater than the timestamp it used previously (so the timestamps used always increase).

$$\forall i, t, t' (T_i(t) \wedge \Diamond(N_i \wedge \Diamond T_i(t')) \Rightarrow t' > t) \tag{Monotonicity}$$

A process knows how to compare known timestamps as described by the following property.

Property 11 If a process j knows it is in the Trying state with a timestamp t' and if j knows that a process i is in the Trying state with a timestamp t and if timestamp t' is less than timestamp t then process j knows that timestamp t' is less than timestamp t

$$\forall j, t', i, t (K_j\,(T_j(t')) \wedge K_j\,(T_i(t)) \wedge [t' < t] \Rightarrow K_j\,[t' < t]) \tag{K-Compare}$$

3.4 Knowledge-Theoretic Mutual Exclusion

Fundamental to distributed mutual exclusion (DME) is the idea that a process j will hold back until another process i is done with the CS.

Definition 1 The proposition that "process j will not enter the CS before a process i with timestamp t is done with the CS" is represented as $blocked(j, i, t)$ and is defined as follows[1]:

$$blocked(j, i, t) \overset{\text{defn}}{\equiv} \neg CS_j\ \hat{\mathcal{W}}\ (\neg CS_j \wedge CS_i(t) \wedge \bigcirc N_i) \tag{Blocked}$$

As a convenience, we will say that process j is *blocked* on process i with timestamp t.

The key to capturing a knowledge-theoretic condition which ensures process i has mutual exclusion using timestamp t is the condition captured in $\phi^{KME}(i, t)$ together with the following two properties:

Definition 2

$$\phi^{KME}(i, t) \overset{\text{defn}}{\equiv} \bigwedge_{j \neq i} K_i\,(blocked(j, i, t)) \tag{ϕ^{KME}}$$

Property 12 For any process i, if i is trying with a timestamp t and and the knowledge guard for mutual exclusion $\phi^{KME}(i, t)$ also holds then at the next point process i will enter its CS with a timestamp t.

$$\forall i, t (T_i(t) \wedge \phi^{KME}(i, t) \Rightarrow \bigcirc CS_i(t)) \tag{CS-Guard}$$

[1]This relatively simple formula disguises considerable subtleties.

Property 13 For any process i, if i is trying with a timestamp t and at the next point process i will enter its CS with a timestamp t then the knowledge guard for mutual exclusion $\phi^{KME}(i,t)$ holds at the current point.

$$\forall i, t(T_i(t) \wedge \bigcirc CS_i(t) \Rightarrow \phi^{KME}(i,t)) \tag{KME}$$

Given only the properties defined so far, we now show that mutual exclusion is guaranteed for the class of DMET systems.

Theorem 1 (Knowledge-Theoretic Mutual Exclusion) *For every point (r,m) of a DMET system, KME implies mutual exclusion, i.e.*

$$\textit{if KME then for all } i,\ t,\ j \neq i,\ t' \ \ (\neg(CS_i(t) \wedge CS_j(t'))) \tag{KMET}$$

Proof: The proof is by contradiction. Suppose for processes i and j that at some point $CS_j(t') \wedge CS_i(t)$. Given the allowable process state transitions (Section 3.2), there exists a point (r,n) such that $T_i(t) \wedge \bigcirc(CS_j(t') \wedge CS_i(t))$. Now by KME, it follows that $(r,n) \models \phi^{KME}(i,t)$.

Using the definition of *blocked*(j,i,t) and noting that $(r,n) \models \bigcirc CS_j(t')$, we have $(r,n) \models \bigcirc(\neg CS_j \,\mathcal{U}(\neg CS_j \wedge CS_i(t) \wedge \bigcirc N_i))$. With $p = \neg CS_j$ and $q = (\neg CS_j \wedge CS_i(t) \wedge \bigcirc N_i)$ and using an expansion for *until* [Manna and Pnueli, 1992] the formula can be rewritten, yielding $(r,n) \models \bigcirc(q \vee [p \wedge \bigcirc(p \,\mathcal{U} q)])$.

Note that both p and q contain $\neg CS_j$ as a conjunct and at the point $(r, n+1)$, we have that $q \vee p$ holds. If q holds then there is a contradiction with $CS_j(t')$, whereas p true also leads to a contradiction with $CS_j(t')$. Hence, mutual exclusion holds. ■

4 DF-ISDME Systems

We are now ready to proceed to a knowledge-theoretic specification for a class of systems corresponding to DF-ISDME protocols. For this paper, we assume that the processes in a DF-ISDME system have *perfect recall* (Section 2.3). In later work we are able to relax this assumption.

4.1 Topology of DF-ISDME Systems

As discussed above, the DF-ISDME protocol is an information structure algorithm restricted to a certain topology involving the *Request* and *Inform* sets. In a message passing environment it is this restricted topology which allows the standard implementation to be *deadlock-free* in the sense of using an efficient exchange of messages [Bonollo and Sonenberg, 1996]. This reliance on message exchange is abstracted away in our specification.

Property 14 We require the Request and Inform sets to satisfy:

$$\forall i \in \mathrm{A}(i \in \mathrm{I}_i) \tag{IINI}$$

$$\forall i \in \mathrm{A}(\mathrm{I}_i \subseteq \mathrm{R}_i) \tag{ISUBR}$$

$$\forall i, j \in \mathrm{A}, i \neq j((i \in (\mathrm{R}_j - \mathrm{I}_j) \wedge j \in (\mathrm{R}_i - \mathrm{I}_i)) \vee (i \in \mathrm{I}_j) \vee (j \in \mathrm{I}_i)) \tag{DF-ISDME}$$

The disjunctive conditions of the DF-ISDME property are usually abbreviated by the following labels: Case I: $i \in (\mathrm{R}_j - \mathrm{I}_j) \wedge j \in (\mathrm{R}_i - \mathrm{I}_i)$; Case II: $i \in \mathrm{I}_j$; or Case III: $j \in \mathrm{I}_i$.

4.2 Protocol for DF-ISDME Systems

The following seven properties involving knowledge-theoretic conditions and actions serve to constrain the behaviour of the protocol being specified. The form of these axioms was suggested by a detailed analysis of the standard DF-ISDME protocol.

Property 15 Whenever a process i enters the Trying state with a timestamp t, it will notify those processes j which are members of its request set R_i ("*i requests j*"):

$$\forall i, t, j \in \mathrm{R}_i(\mathit{enter}(T_i(t)) \Rightarrow \mathit{notify}_{ij}(T_i(t))) \tag{Request}$$

Property 16 For any process j that is a member of i's request set, if j is in the Noncritical state and if j knows that process i is trying with timestamp t then j will notify i that j will not enter the CS before i with t ("*j grants* to i" since j isn't trying to enter the CS at present).

$$\forall i, t, j \in \mathrm{R}_i(N_j \wedge K_j\,(T_i(t)) \Rightarrow \mathit{notify}_{ji}(\mathit{blocked}(j, i, t))) \tag{Noncritical Grant}$$

Property 17 For any process j that is a member of i's request set, if j is in its CS with a timestamp t' and at the next point j is in the Noncritical state and if j knows that process i is trying with timestamp t then j will notify i that j will not enter the CS before i with t ("*j grants* to i" since j is about to exit the CS).

$$\forall i, t, t', j \in \mathrm{R}_i(CS_j(t') \wedge \bigcirc N_j \wedge K_j\,(T_i(t)) \Rightarrow \mathit{notify}_{ji}(\mathit{blocked}(j, i, t))) \tag{Last CS Grant}$$

Property 18 For any process i that is a member of j's inform set, whenever j is exiting the CS with timestamp t', j will notify i that it will never again enter the CS with timestamp t' ("*j releases the CS for i*").

$$\forall j, t', i \in \mathrm{I}_j(CS_j(t') \wedge \bigcirc N_j \Rightarrow \mathit{notify}_{ji}(\text{⬘}(CS_j(t') \wedge \bigcirc N_j))) \tag{Release}$$

Property 19 For any process j that is a member of i's request set, if j is in the Trying state with its associated timestamp t' and where j knows that a process i is also trying with a timestamp t and given that there is a losing arbitration for j with t' compared to i with t then j will notify i that j will not enter the CS before i with t ("*j grants* to i" since j is the loser in the arbitration with i).

$$\forall i, t, t', j \in \mathrm{R}_i(T_j(t') \wedge K_j\,(T_i(t)) \wedge [t' > t] \Rightarrow \mathit{notify}_{ji}(\mathit{blocked}(j, i, t))) \tag{Lose}$$

Property 20 For any process j such that a process i is a member of j's request set, if i is trying with timestamp t and where i knows that process j is trying with timestamp t' and given that there is a winning arbitration for i with t compared to j with t' then i knows that j will not enter the CS before i with t (here i is the winner in the arbitration with j).

$$\forall j, t', t, i \in \mathrm{R}_j(T_i(t) \wedge K_i\,(T_j(t')) \wedge [t < t'] \Rightarrow K_i\,(\mathit{blocked}(j, i, t))) \tag{Win}$$

Property 21 For any process j such that a process i is a member of j's inform set, if i is trying with timestamp t and where i doesn't know that process j is trying with timestamp t' and given that the following two conditions hold:

1. for any other process k not equal to i,j that is a member of i's request set, i knows that k will not enter the CS before i with t;

2. for any other process k not equal to i,j such that i is a member of k's inform set, if i knows that process k is trying with timestamp t'' then i knows that k will not enter the CS before i with t;

then i knows that j will not enter the CS before i with t (here i has enough knowledge from the other processes k to assert its right to enter the CS despite not knowing about j).

$$\begin{aligned}&\forall j, t', t, i \in \mathrm{I}_j(\\ &\qquad [T_i(t) \wedge \neg K_i\,(T_j(t')) \wedge \forall k \neq i, j,\ k \in \mathrm{R}_i(K_i\,(\mathit{blocked}(k, i, t))) \wedge \\ &\qquad\qquad \forall k \neq i, j,\ i \in \mathrm{I}_k(K_i\,(T_k(t'')) \Rightarrow K_i\,(\mathit{blocked}(k, i, t)))] \\ &\qquad\qquad\qquad \Rightarrow K_i\,(\mathit{blocked}(j, i, t))) \quad \text{(Dictate)}\end{aligned}$$

4.3 Knowledge-Theoretic DF-ISDME Memory Lemmas

In this section, we derive knowledge-theoretic DF-ISDME memory lemmas which highlight, for a sequence of points in a run, the relationships between processes and their mutually allowable process states. Apart from the description and proof for Lemma 3, which is typical, we omit the descriptions and proofs of the other memory lemmas from this section. We also omit all corollaries of the other memory lemmas in this section, but we note that their knowledge-theoretic form is similar to the form of Corollary 1.

Lemma 1

$$\forall i, t, j, t'(\mathit{CS}_j(t')\,\mathcal{S}(\mathit{CS}_j(t') \wedge T_i(t)) \Rightarrow T_i(t))$$

Lemma 2

$$\forall j, t', t, i \in \mathrm{R}_j((T_j(t') \vee \mathit{CS}_j(t'))\,\mathcal{S}(T_j(t') \wedge K_j\,(T_i(t)) \wedge [t' < t]) \Rightarrow T_i(t))))$$

Lemma 3 *The following proposition is valid in the class of DF-ISDME systems: If a process i is still in the Trying state with a timestamp t since a point in the past when a process j was blocked on process i with timestamp t then process j is still blocked on process i with timestamp t.*

$$\forall i, t, j(T_i(t)\,\mathcal{S}\,\mathit{blocked}(j, i, t) \Rightarrow \mathit{blocked}(j, i, t))$$

Proof: Assume $T_i(t)\,\mathcal{S}\,\mathit{blocked}(j, i, t)$ is true at some point (r, v). Then from the definition of the $\mathcal{S}$ operator, there exists a point in the past (r, n), $n \leq v$, such that $\mathit{blocked}(j, i, t)$ holds. Now if $n = v$ then $\mathit{blocked}(j, i, t)$ follows directly. □

In the case where $n < v$, it follows that $(r, n + 1..v) \models T_i(t)$ holds. Now since $(r, n) \models \mathit{blocked}(j, i, t)$, it follows that there are two cases arising from the definition of *blocked*. In the first case, $(r, n + 1) \models \Box\neg \mathit{CS}_j$ holds. Since $\Box\neg \mathit{CS}_j$ holds forever from $(r, n + 1)$ and since $n < v$, it follows that $(r, v) \models \mathit{blocked}(j, i, t))$. □

In the second case, $(r, n + 1) \models \neg \mathit{CS}_j\ \mathcal{U}\ (\neg \mathit{CS}_j \wedge \mathit{CS}_i(t) \wedge \bigcirc N_i)$ holds. It follows from the definition of the $\mathcal{U}$ operator, that there exists the earliest point (r, s), $s \geq n + 1$, such that $(r, s) \models \neg \mathit{CS}_j \wedge \mathit{CS}_i(t) \wedge \bigcirc N_i$ holds. This means that $\neg \mathit{CS}_j$ holds from $(r, n + 1)$ through to (r, s) inclusive. Therefore, it follows that, in this case, $\mathit{blocked}(j, i, t)$ holds from (r, n) through to $(r, s - 1)$.

Given $\mathit{CS}_i(t)$ at the point (r, s) and given TC, there must exist the point (r, u) which is the last point where $T_i(t)$ holds prior to $\mathit{CS}_i(t)$ holding at $(r, u + 1)$. Now, using the fact that $(r, n + 1..v) \models T_i(t_i)$ must hold, it must be that case that $n + 1 \leq v \leq u$. From this, it follows that the interval $(r, n + 1..u)$ lies within the interval $(r, n..s - 1)$ where $\mathit{blocked}(j, i, t)$ holds. Hence, $\mathit{blocked}(j, i, t)$ holds at the point (r, v). ■

The following corollary follows immediately from validity in S5:

Corollary 1 *The following proposition is valid in the class of DF-ISDME systems:*

$$\forall i,t,j(K_i\,(T_i(t)\,\mathcal{S}\,blocked(j,i,t))\Rightarrow K_i\,(blocked(j,i,t)))$$

Lemma 4

$$\forall i,t,j,t'(T_i(t)\,\mathcal{S}(blocked(j,i,t)\wedge T_j(t'))\Rightarrow blocked(j,i,t)\wedge T_j(t'))$$

Lemma 5

$$\forall i,t,j,t'((T_i(t)\wedge\neg\diamondminus(CS_j(t')\wedge\bigcirc N_j))\,\mathcal{S}(T_i(t)\wedge blocked(i,j,t'))\Rightarrow blocked(i,j,t'))$$

Lemma 6

$$\forall i,t,j,t'(\diamondminus(CS_j(t')\wedge\bigcirc N_j)\Rightarrow\Box(\neg blocked(i,j,t')\wedge\neg T_j(t')))$$

5 Liveness

We are now ready to establish the liveness result for DF-ISDME systems that any process i which is trying to enter the critical section will eventually do so, i.e.

$$\forall i,t(T_i(t)\Rightarrow\Diamond CS_i(t))$$

This can be seen to follow immediately from the CS-Guard property (Property 12, which describes a guard condition for entry to the critical section) and a liveness result for the guard condition (**Theorem 2**) which establishes that a trying process will eventually achieve the guard condition. This last is the main result of the paper.

Theorem 2 (Knowledge-Theoretic DF-ISDME Liveness) *For any given point in a run (r,m) of a DF-ISDME system, if process i is trying to enter its CS with timestamp t, denoted $T_i(t)$, then eventually $\phi^{KME}(i,t)$ holds:*

$$\forall i,t(T_i(t)\Rightarrow\Diamond\phi^{KME}(i,t)) \qquad \text{(K-Live)}$$

Proof: The full proof runs to several pages so we provide only an outline here: We fix a run r of a DF-ISDME system and consider a series of points in r at which key conditions crucial to our liveness argument hold. These conditions are summarised in Figure 1 for reference. Overall, the proof proceeds in four stages as follows: In Stage 1, we assume, by way of contradiction, that the set of *halted* processes contains those processes which, in a run r of a DF-ISDME system, eventually try forever but *never* enter their respective CS. We choose a timestamp t that is the *minimum* timestamp from the set of timestamps associated with this set of halted processes. Following on from this, we choose the point (r,m) as the *earliest* such point in the run r such that the condition that i is trying forever with timestamp t holds true.

In Stage 2, we consider all processes j that have a Case I or a Case III DF-ISDME topology relationship with respect to process i which is trying with t. We show that there exists a point (r,m'), $m'\geq m$, such that i *knows* that all these processes j are *blocked* on i trying with timestamp t.

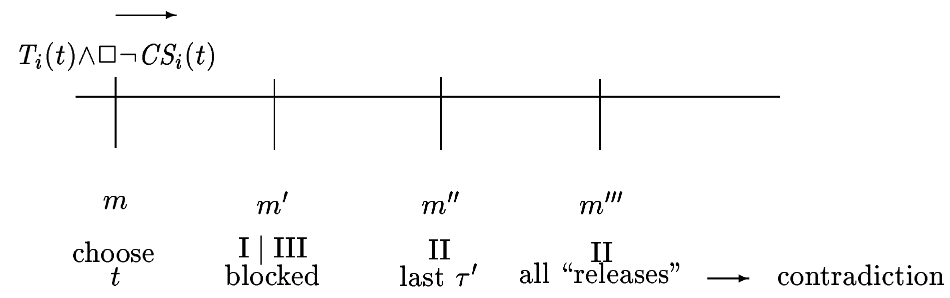

Figure 1: Liveness proof - key points in run r of a DF-ISDME system.

For Stage 3 of the proof, we consider the remaining topology case, i.e. all processes j that have a Case II DF-ISDME topology relationship with respect to process i which is trying with t. Our argument proceeds in two parts: For the first part, we show that there exists a point (r, m''), $m'' \geq m'$, where all these processes j have tried, or are trying, with their *last* generated timestamp τ'. This requires us to show that the set of timestamps t' *generated* by the Case II processes j is *finite*. For the second part, we show that there exists a point (r, m'''), $m''' \geq m''$, where i knows that all Case II processes j, which had been granted precedence to enter the CS ahead of i, have now released the CS.

In Stage 4, the final stage of the proof, we show that at the point (r, m'''), arrived at in Stage 3, i *knows* that all the Case II processes j are *blocked* waiting on i trying with timestamp t by virtue of the Win or Dictate properties. This proposition, together with the proposition established in Stage 2 at the point (r, m'), means that i will achieve the knowledge-theoretic guard condition $\phi^{KME}(i, t)$ and hence enter its CS. This is in contradiction with what was assumed initially, that i was a member of the set of halted processes, hence the theorem is proved.

The following key observations should be noted concerning the liveness proof. The *monotonicity* of the timestamps generated in a run is used repeatedly. In particular, monotonicity is used to show the *finiteness* of the set of candidates which have precedence over t. The careful choice of t associated with the concept of the *minimal halted process* is used to ensure: that the Case I or Case III processes j whose timestamps have precedence over t eventually exit their CS (Stage 2); and that the Case II processes j whose timestamps have precedence over t and receive "grants" from i will also eventually exit their CS (Stage 3). As a further remark, we note that the proof makes use of the memory lemmas defined in Section 4.3 together with repeated application of the perfect recall properties (Section 2.3). ■

6 Conclusion

Our knowledge-theoretic specification allows us to gain a deeper understanding of pragmatic classes of algorithms such as DF-ISDME. For example, the Dictate axiom is the most complex axiom in our specification—and yet its behavioural impact isn't obvious at all when looking at a standard DF-ISDME algorithm implementation [Bonollo and Sonenberg, 1996]. Given such an implementation of processes in a message passing environment, the Dictate axiom allows a process holding a lower priority timestamp to sometimes "jump ahead" of a higher priority process when messages from the higher priority process are delayed in transit—but this behaviour isn't mentioned at all in the standard algorithm proofs in the conventional literature. Instead, the standard algorithm proofs tend to imply (by omission) that all requesting processes will enter the CS in timestamp order. Interestingly, the conjuncts of the

Dictate axiom premise are very similar to the "while loop test condition" of the standard DF-ISDME algorithm. In the standard DF-ISDME algorithm, a process waits for this test condition to become true to decide when mutual exclusion has been achieved.

This deeper understanding of DF-ISDME algorithms can be exploited in a practical way. For DF-ISDME, the knowledge-theoretic specification leads to plug-ins of different implementations. For example, our specification highlights the fact that monotonicity is the key requirement for timestamp implementations. Thus, unlike many standard algorithms, we can easily claim that Lamport timestamps [Lamport, 1978] are not necessary to implementations of our specification. Hence, for example, a simple integer count of the number of times a CS has been visited will suffice.

The knowledge-theoretic approach allows us to "abstract out" the environment in various ways. For example, during the proof of our main result for liveness, we discovered and applied several memory lemmas (Section 4.3). Whenever a memory lemma was applied in the proof, it was followed by the application of a perfect recall property (Section 2.3). Thus using this method, we have collected a set of process memory usage requirements for the DF-ISDME class of algorithms.

In future work, we plan to recast the specification by modifying the current memory lemmas so as to make the memory assumptions used explicit whilst dropping the global perfect recall assumption. This will also allow us to rewrite the specification beyond the conventional knowledge-theoretic approach using new developments based on a *logic of local propositions* [Engelhardt *et al.*, 1998].

References

[Bonollo and Sonenberg, 1996] U. Bonollo and E.A. Sonenberg. Deadlock-free information structure distributed mutual exclusion algorithms. In *Proceedings of the 19th ACSC Conference*, Melbourne, Australia, January 31–February 2 1996.

[Engelhardt *et al.*, 1998] K. Engelhardt, R. van der Meyden, and Y. Moses. Knowledge and the logic of local propositions. In *Conf. on Theoretical Aspects of Rationality and Knowledge*, Evanston, Il, July 1998.

[Fagin *et al.*, 1995] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning About Knowledge.* The MIT Press, Cambridge, Massachusetts, 1995.

[Halpern and Zuck, 1992] J.Y. Halpern and L.D. Zuck. A little knowledge goes a long way: Knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the Association for Computing Machinery*, 39(3):449–478, 1992.

[Halpern *et al.*, 1997] J.Y. Halpern, R. van der Meyden, and M.Y. Vardi. Complete axiomatizations for reasoning about knowledge and time [submitted for publication]. 1997.

[Lamport, 1978] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, pages 558–564, July 1978.

[Manna and Pnueli, 1992] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1992.

[Moses and Kislev, 1993] Y. Moses and O. Kislev. Knowledge-oriented programming. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pages 261–270, 1993.

[Moses, 1997] Y. Moses. [private communication]. 1997.

[Sanders, 1987] B.A. Sanders. The information structure of distributed mutual exclusion algorithms. *ACM Transactions on Computer Systems*, 5(3):284–299, 1987.

[Singhal, 1993] M. Singhal. A taxonomy of distributed mutual exclusion. *Journal of Parallel and Distributed Computing*, 18:94–101, 1993.

[Vardi, 1996] Moshe Y. Vardi. Implementing knowledge-based programs. In Yoav Shoham, editor, *Theoretical Aspects of Rationality and Knowledge: Proceedings of the Sixth Conference (TARK 1996)*, pages 15–30. Morgan Kaufmann, San Francisco, 1996.