Detection of Global Predicates: Techniques and their Limitations*

Craig M. Chase† Vijay K. Garg‡

Parallel and Distributed Systems Laboratory
email: pdslab@ece.utexas.edu
Electrical and Computer Engineering Department
The University of Texas at Austin,
Austin, TX 78712
http://maple.ece.utexas.edu/

Abstract

We show that the problem of predicate detection in distributed systems is NP-complete. In the past, efficient algorithms have been developed for special classes of predicates such as stable predicates, observer-independent predicates, and conjunctive predicates. We introduce a class of predicates, *semi-linear predicates*, which properly contains all of the above classes. We first discuss stable, observer-independent and semi-linear classes of predicates and their relationships with each other. We also study closure properties of these classes with respect to conjunction and disjunction. Finally, we discuss algorithms for detection of predicates in these classes. We provide a non-deterministic, detection algorithm for each class of predicate. We show that each class can be equivalently characterized by the degree of non-determinism present in the algorithm. Stable predicates are defined as those that can be detected by an algorithm with the most non-determinism. All other classes can be derived by appropriately constraining the non-determinism in this algorithm.

keywords: distributed debugging, predicate detection, unstable predicates.

* A preliminary version of this paper appeared in the Ninth Workshop on Distributed Algorithms, Le Mont Saint Michel, France, September 1995, pp 303–317.

† supported in part by NSF Grant CCR-9409736

‡ supported in part by the NSF Grants ECS-9414780, CCR-9520540, a General Motors Fellowship, and an IBM grant

1 Introduction

Detection of a global predicate is a fundamental problem in distributed computing. This problem arises in many contexts such as designing, testing and debugging of distributed programs. For example, the detection of global predicate arises in implementing the most basic command of a debugging system:"stop the program when the predicate ϕ is true." To stop the program, it is necessary to detect the predicate ϕ; a non-trivial task if ϕ requires access to the global state.

There are three approaches for detecting global predicates. The first approach is based on the global snapshot algorithm by Chandy and Lamport [4, 3, 24]. Their approach requires repeated computation of consistent global snapshots until a snapshot is found in which the desired predicate is true. This approach works only for stable predicates, *i.e.* predicates that do not turn false once they become true. If the desired predicate ϕ were not stable then their approach may fail to detect the predicate because ϕ may turn true only between two successive snapshots.

The second approach to global predicate detection is based on the construction of the lattice of global states [7, 17, 20]. This approach, first presented by Cooper and Marzullo [7], allows the detection of *definitely:*ϕ and *possibly:*ϕ . The predicate *possibly:*ϕ is true if ϕ is true for any global state in the lattice. The predicate *definitely:*ϕ is true if, for all paths from the initial global state to the final global state, ϕ is true in some global state along that path. This approach can detect both stable and unstable predicates. However, detection may be prohibitively expensive. In a system with n processes each with m "relevant" local states, this approach requires exploring $O(m^n)$ global states in the worst case.

The third approach is based on exploiting the structure of the predicate ϕ. This approach is less general than the second, but results in more efficient detection algorithms. For example, [13, 12] present algorithms to detect *possibly:*ϕ and *definitely:*ϕ of complexity $O(n^2m)$ when ϕ is a conjunction of local predicates. Similarly, [15], and independently [26, 6] present efficient algorithms to detect $\sum x_i < C$ where the x_i are variables on different processes and C is constant. In [25], Stoller and Schneider propose a hybrid of the second and third approaches that reduces the size of the lattice that must be explored during detection. This method may require exponential complexity to detect some predicates.[1]

The best approach to use in a given application depends upon the specific predicate that is to be detected. For example, a snapshot algorithm may be acceptable to detect termination, but cannot be used to detect violations of mutual exclusion. In [5], predicates are classified into three types: stable, observer-independent, and

[1] Stoller and Schneider's method is exponential in the size of the *fixed set*. The cardinality of the fixed set is at most $n - 1$.

general. Membership of a predicate in a class can be determined by its truthness in different modalities. A stable predicate is one for which ϕ is true in a global state whenever *possibly:*ϕ is also true in that state. An observer-independent predicate is one for which *definitely:*ϕ and *possibly:*ϕ are equivalent.

The contributions of this paper are:

- We show that the third approach cannot always yield an efficient detection algorithm. In particular, the problem of detecting whether a boolean expression became true in a distributed computation is NP-complete. The problem stays NP-complete even when processes do not communicate with each other and each process executes a single instruction.

- We define the *linear* class of predicates. We show that the set of global states satisfying a predicate ϕ is an inf-semilattice if and only if ϕ is a linear predicate. Thus, linearity captures the class of predicates for which the first satisfying global state exists. We also present an algorithm to detect the first global state for which linear ϕ is true.

- By considering the dual predicate of linearity, we get a necessary and sufficient condition for a given set of global states to be a lattice. This generalizes many earlier results. For example, the fact that the set of all recoverable states form a lattice [18] is an easy consequence of our result. Similarly, the monotonicity condition on channel predicates [11] is also a special case of linearity.

- We define a larger class of predicates, *semi-linear* that includes linear, observer-independent and stable predicates as proper subclasses. We describe the closure properties of all four classes under conjunction and disjunction.

- We provide a family of non-deterministic detection algorithms. Beginning with an algorithm to detect stable predicates, we show that by constraining the non-determinism we can derive algorithms that detect predicates in the other classes. We show that each algorithm will detect precisely those predicates that are members of the corresponding class. Thus, the degree and type of non-determinism in a detection algorithm provides an equivalent means to classify predicates.

- We generalize the definitions for *possibly:*ϕ and *definitely:*ϕ for ranges of execution. In practice, it is often not desirable or even possible to begin monitoring a predicate at the start of execution. Thus, one would like to determine if ϕ became true between two consistent global states. Our algorithms detect

Table 1: Notation used in this paper

$P \equiv \{p_1 \ldots p_n\}$	the set of processes in the system
s, t	a local state
$s \rightarrow t$	s happens before t
W, X, Y, Z	a consistent cut
$X \leq Y$	Y is reachable from X
X^s	The cut formed by advancing X to the successor of s
$S \equiv \{X_1, \ldots X_k\}$	a sequence of cuts
$\phi(X)$	The predicate ϕ evaluated in X
$P_\phi(X, Y)$	The predicate *possibly:*ϕ is true between X and Y
$D_\phi(X, Y)$	The predicate *definitely:*ϕ is true between X and Y

possibly:ϕ given two bounding global states. Alternatively, the upper bound can be left unspecified, allowing the algorithms to be used on-line.

In this paper, we restrict ϕ to be a condition defined on the values of program variables in a single global state. Other work in predicate detection has considered predicates defined on sequences of states. For example, [22, 12] discuss linked predicates, [16, 2] discuss atomic sequences, and [9] discuss regular patterns. We also refer the reader to [1, 23, 10] for surveys of stable and unstable predicate detection.

This paper is organized as follows. In Section 2 we describe our model of a distributed system, and introduce the terminology used. In Section 3 we show that detection of possibly ϕ is NP-Complete. Section 4 defines the predicate classes (stable, observer-independent, linear). Examples of predicates, the closure properties, and significance of each class are described. Section 5 presents the detection algorithms.

2 Our Model of the Execution of a Distributed Program

Table 1 summarizes the notation used in this paper. A distributed system consists of a set of processes $P \stackrel{\text{def}}{=} \{p_1, \ldots, p_n\}$. Each process executes a predefined program. Processes do not share any clock or memory; they communicate and synchronize with each other by sending messages over a set of channels. We assume that messages are not lost, altered, or spuriously introduced into a channel. We do not assume that channels are FIFO.

We model the execution of each process in the distributed system as a sequence of distinct local states. We use lowercase letters s and t to represent local states.

Following Lamport [19] we define the causally-precedes relation $\rightarrow$ (also known as "happened before") as follows:

$s \rightarrow t$ if and only if:

1. s and t are states from the same process, and s precedes t in the execution of that process.
2. s and t are states from different processes and a message is sent in state s and received in state t.
3. there exists some state u such that $s \rightarrow u$ and $u \rightarrow t$.

Two states are said to be *concurrent* (denoted $s \| t$) when neither state happened before the other. Formally:

$$s \| t \stackrel{\text{def}}{=} s \not\rightarrow t \wedge t \not\rightarrow s$$

We use the term *local state* to refer to a state s from a single process. A *global state* is a set of concurrent local states, one from each process. We require that the execution of a distributed system commence with a global state. A *cut* is a partial execution of the program. It must end with a global state. Formally, a cut is a downset of local states (ordered by $\rightarrow$), which includes exactly n concurrent supremal states. We use symbols X, Y and Z to denote cuts.

Given a cut, X, we use the notation $X \leq Y$ to denote that Y is a cut reachable from X. That is, $X \subseteq Y$ and there is an execution of the system that takes it from X to Y. [2] Note that the set of states from a single process is a total order. That is, each s has a well-defined successor, denoted by succ(s) (unless s is the final state in the process). When s is a supremal state in cut X, we denote by X^s the cut formed by including in X the immediate successor to s. Formally

$$X^s \stackrel{\text{def}}{=} X \cup \{\text{succ}(s)\}$$

It is well known that the set of global states from an execution form a lattice [21]. In our terminology the equivalent result is: given a cut Y, the set of all cuts $\{\text{X} : X \leq Y\}$ forms a lattice with respect to $\leq$.

A predicate is a boolean-valued function whose domain is the set of all possible cuts from all possible executions of a distributed system. The predicate detection problem is concerned with identifying a cut (or cuts) in which a predicate evaluates to true. We use ϕ to denote the predicate of interest. We assume in this paper that $\phi(X)$ is easy to compute given X. The difficulty in the predicate detection problem

[2] Note that since cuts are downsets, $X \subseteq Y$ implies X is a prefix of Y.

is attributable to the fact that the number of cuts from any execution is exponential in the number of processes.

We use the notation $X[i]$ to refer to the supremal local state, s, in X from process p_i.

Finally, we use S to denote a sequence of cuts; each cut in the sequence reachable from the previous cut. That is, $S = \{X_1, \ldots X_k\}$ where for all i, $X_i < X_{i+1}$. A sequence of cuts is called a *path* iff for all i, X_i and X_{i+1} differ by exactly one local state. The term "observation" is used to describe a path in [5].

3 NP-Completeness of Global Predicate Detection

The global predicate detection problem (GLOB) is a decision problem. It takes the form of:

> **Given**: an execution Y of n processes, an initial cut $X \leq Y$, and a predicate ϕ.
> **Determine** if there exists a cut $W : X \leq W \leq Y$ such that $\phi(W)$ is true.

We show that the predicate detection problem is NP-Complete.

Theorem 3.1 *GLOB is NP-complete.* **Proof:** *First note that the problem is in NP. A verifier for the problem takes as input a cut W and then determines if the predicate is true. Therefore, if $\phi(W)$ can be evaluated in polynomial time, then the detection of ϕ belongs to the class NP.*

We show NP-completeness of the simplified predicate detection problem where ϕ is a boolean function of a set of program variables, $\{u_1, \ldots u_n\}$. Each of the n processes contributes one program variable. Furthermore, program variables are restricted to taking the values "true" or "false". We reduce the satisfiability problem (SAT) to GLOB by constructing an appropriate execution.

*In an instance of SAT, we are given a boolean expression, $e(u_1, \ldots u_n)$, and we wish to determine if there exist a set of truth values for the u_i such that e evaluates to true. To answer this question, we construct a distributed system with n processes such that each u_i is a variable in process p_i. The execution Y consists of two local states from each p_i. In the first state, u_i has the value false. In the second state, u_i has the value true. There are no messages exchanged during the computation. The initial cut X is the initial global state of the system (*i.e. *X consists of the first state from each process). The value of predicate ϕ is the result of evaluating e on the variables in the system.*

It is easily verified that the predicate ϕ is true for some cut between X and Y if and only if the expression is satisfiable. 2

The above result suggests that detection of a general global predicate is intractable even for simple distributed computation (boolean variables, no messages exchanged). However, many predicates are known to be detectable in polynomial time. The remaining sections discuss classes of predicates for which efficient detection algorithms are known.

4 Classes of Predicates

In this section we describe four classes of predicates; stable, observer-independent, linear and semi-linear. First, we extend the predicate modalities defined by Cooper and Marzullo in [8]. We define possibly ϕ to hold between cuts X and Y if there exists at least one cut, W, $X \leq W \leq Y$ such that ϕ is true in cut W. We denote this as $P_\phi(X, Y)$. Formally:

$$P_\phi(X, Y) \stackrel{\text{def}}{=} \exists W : X \leq W \leq Y : \phi(X)$$

Definitely ϕ is defined to hold between X and Y iff every path from X to Y includes at least one cut for which ϕ is true. We use the notation $D_\phi(X, Y)$ defined as follows:

$$D_\phi(X, Y) \stackrel{\text{def}}{=} \forall S : \text{S is a path from X to Y} : (\exists W \in S :: \phi(W))$$

For completeness, when $X \not\leq Y$ we define $P_\phi(X, Y)$ and $D_\phi(X, Y)$ to be false. Obviously:

$$\begin{aligned} \phi(X) \vee \phi(Y) &\Rightarrow D_\phi(X, Y) \\ D_\phi(X, Y) &\Rightarrow P_\phi(X, Y) \end{aligned}$$

4.1 Stable Predicates

The best known classification of predicates is that of the *stable* and the *unstable* predicates. Simply put, a stable predicate remains true once it becomes true. More formally, we say that ϕ is stable if and only if:

$$\forall X, Y : X \leq Y : \phi(X) \Rightarrow \phi(Y)$$

Well known examples of stable predicates are, termination, and garbage collection. It must be noted that stability depends on the system. Some properties are stable in some systems but not stable in others. For example, the formula,

$$\min_i x_i > k$$

is a stable predicate in distributed simulation environments when the x_i are the timestamps in every process and every message. Clearly, it is not stable for arbitrary systems. Recall that a *predicate* in our terminology includes not just the formula, but the system as well.

4.2 Observer Independent Predicates

Charron-Bost *et al* [5] describe a class of predicates that they call "observer independent". In our notation, this class includes all predicates such that $\forall X, Y : P_\phi(X,Y) \Leftrightarrow D_\phi(X,Y)$. The name observer independent stems from the notion of a set of observers where each witnesses a different sequential execution of the system. Each observer can determine if ϕ became true in any of the cuts witnessed by them. If the predicate is observer independent, then all observers will agree on whether ϕ ever became true. It must be noted that our definition of observer independent is stronger than that given in [5]. They defined possibly and definitely ϕ with respect to the initial state of the system. Since we require possibly ϕ and definitely ϕ to be equivalent for any range of cuts, some predicates which would be observer independent in their definition are not observer independent with ours.

Any stable predicate is also observer independent. A proof of this fact appears in [5]. In our model, the proof takes the following form: given a stable predicate ϕ,

$$\forall X, Y : P_\phi(X,Y) \Rightarrow \phi(Y)$$

That is, if a stable predicate ever becomes true in a cut that precedes Y, then it must still be true in Y. Further, since $\phi(Y) \Rightarrow D_\phi(X,Y)$ for any X, it follows that $P_\phi(X,Y) \Rightarrow D_\phi(X,Y)$. Recall that for any predicate, $D_\phi(X,Y) \Rightarrow P_\phi(X,Y)$. Thus, when ϕ is stable, possibly and definitely ϕ are equivalent.

An example of an observer-independent predicate is a disjunction of local predicates. Consider a distributed system with predicate b defined on one process, and predicate c defined on another. Then the predicate $\phi = b \vee c$ is observer independent [5].

4.3 Linear Predicates

In this paper we introduce two new classes of predicates, *linear* and *semi-linear*. A predicate belonging to either of these classes can be detected efficiently (see Section 5). Linear predicates subsume conjunctive predicates and channel predicates [13, 14]. A linear predicate is based on the definition of a "forbidden" state. States are forbidden with respect to a predicate, ϕ, and a cut X.

$$\text{forb}_\phi(s, X) \stackrel{\text{def}}{=} \forall Y : X \leq Y : \neg\phi(Y) \vee X^s \leq Y$$

The intuition of $\text{forb}_\phi(s, X)$ is that ϕ must remain false until a successor to s is reached. We say that a predicate is *linear* if for any cut X in which the predicate is false, at least one of the supremal states in X is forbidden.

$$\text{linear}(\phi) \stackrel{\text{def}}{=} \forall X :: \neg\phi(X) \Rightarrow (\exists s \in \sup(X) :: \text{forb}_\phi(s, X))$$

An important subclass of linear predicates are conjunctive predicates [13], *e.g.* $b < 10 \wedge c = 3$ where b and c are variables on different processes. To evaluate this predicate in a cut, we use the values of b and c from the last state of the respective processes in that cut. Any conjunctive predicate is linear. Consider a cut, X, in which a conjunction of local predicates is false. From the definition of a conjunctive predicate, there must exist at least one supremal state, s, for which the corresponding local predicate is false. This state clearly satisfies the definition of being forbidden in X.

Some boolean-valued functions are linear in some systems but not in others. For example, consider the formula "$b + c < k$" for real-valued variables b and c on different processes and constant k. This formula defines a linear predicate for systems in which either b or c is monotonically increasing. For example, if b is monotonically increasing, then in any cut where predicate is false, the supremal state with variable c is forbidden. A new value for c must be found before the predicate can become true, since b can only increase in value.

Many interesting channel predicates [14] also satisfy linearity. A channel predicate is defined as a boolean function on the state of a uni-directional channel. Note that a channel predicate is not a local predicate; it depends on the state of the sender and the receiver. Consider the channel predicate, "there are no messages in the channel." This channel predicate may be useful in detecting termination of a computation. We show that this channel predicate is linear. If the predicate is false in some cut, then the supremal state corresponding to the receiver of the channel is forbidden. This is because a non-empty channel can become empty only by receiving messages. Thus, until at least one new state is reached by the receiving process, the predicate will remain false regardless of action taken by other processes. More generally, the predicate "there are exactly k messages in the channel" is linear.

Linear predicates are not necessarily stable. Any conjunctive predicate is a counter example. If at least one of the local predicates is unstable, then the conjunction of these properties is also unstable. Furthermore, conjunctive predicates are not necessarily observer independent. Consider the simple formula $\phi = a \wedge b$, for boolean variables a and b on two different processes. Figure 1 shows a lattice of cuts in which $P_\phi(X_1, X)$ is true yet $D_\phi(X_1, X)$ is not. The predicate does not hold in X_1. There are two possible paths from X_1 to X. Along the path $\{X_1, X_2, X\}$, ϕ is always false. However, the path $\{X_1, X_3, X\}$ does include a cut, X_3, for which ϕ is true.

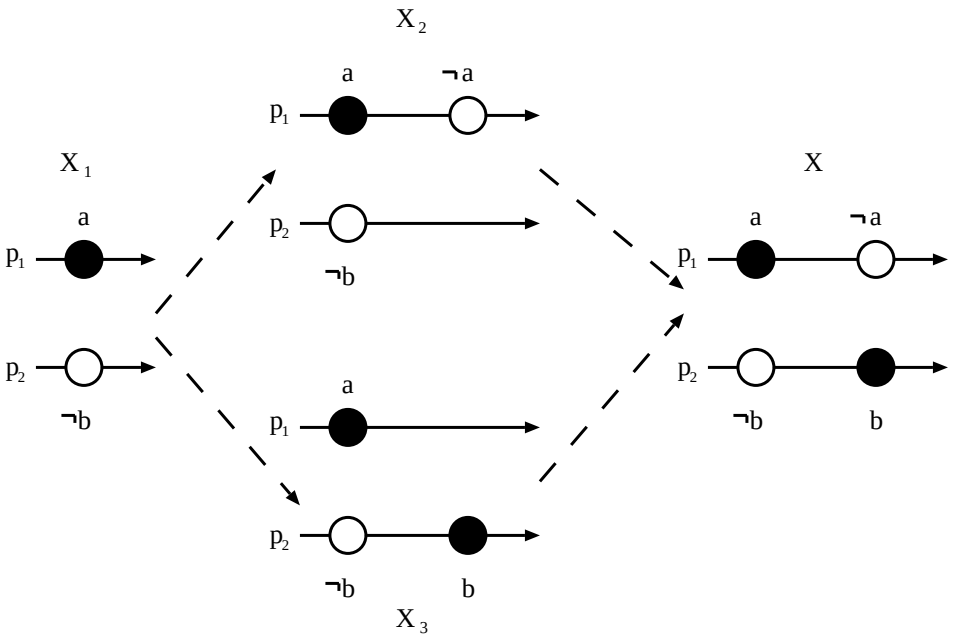


Figure 1: Conjunctive predicates may not be observer independent.

The class of observer independent predicates is not contained within the class of linear predicates. For example, a disjunction of local predicates is observer independent, but not linear (see Section 4.5.4).

4.4 Semi-Linear Predicates

The semi-linear class of predicates contains all three of the previous classes (stable, observer-independent, and linear). Semi-linear predicates are also relatively easy to detect. The class is defined with respect to the semi-forbidden property of states and cuts. We say that state s is semi-forbidden in cut X if it satisfies the following definition:

$$\text{sforb}_\phi(s, X) \stackrel{\text{def}}{=} \forall Y : X^s \leq Y : P_\phi(X, Y) \Rightarrow P_\phi(X^s, Y)$$

That is, Y indicates any possible continuation of the system (after X) that includes at least the immediate successor to s. If any of the new states from Y permit the predicate to become true (*i.e.* $P_\phi(X, Y)$), then it must also be true in some cut that is strictly greater than X and includes the immediate successor to s. The intuition of semi-forbidden is that s is irrelevant to the truth-value of the predicate. While looking for a cut where the predicate is true, we can disregard s in favor of its successor.

Since we consider only consistent cuts in this paper, we require the state that is semi-forbidden to be an *eligible* state. Let p be a process in the system, and let s be the supremal state from p in some cut X. Note that advancing the cut along p may not be possible for two reasons. First, s may be the final state from process p. Second, even if s has a successor, t, then $X \cup \{t\}$ may not be a consistent cut. That is, p receives a message between s and t, and the state where that message was sent is not part of X. For any consistent cut X, we use $\text{eligible}(X)$ to denote the set of states along which X can be advanced while maintaining consistency. Formally,

$$\text{eligible}(X) \stackrel{\text{def}}{=} \{s \in \text{sup}(X) | \neg\text{final}(s) \wedge \forall t \in \text{sup}(X) : t \not\rightarrow \text{succ}(s)\}$$

Note that in our model, the eligible set for X is empty if and only if X is a complete cut. That is, the system has executed to termination in X. We denote this condition by $\text{final}(X)$.

Lemma 4.1

1. *Let X and Y be two consistent cuts such that $X \leq Y$ and $\exists s \in eligible(X) : s \in Y$. Then, Y^s is also consistent.*

2. *Let X and Y be two consistent cuts such that $X < Y$ and eligible(X) is a singleton $\{s\}$. Then, $X^s \leq Y$.*

Proof: *Follows from the definition of eligibility.* 2

We now define the semi-linear class of predicates:

$$\text{semi-linear}(\phi) \stackrel{\text{def}}{=} \forall X : [\neg\phi(X) \wedge \neg\text{final}(X) \Rightarrow \exists s \in \text{eligible}(X) : \text{sforb}_\phi(s, X)]$$

We first show that the class of semi-linear predicates strictly includes the class of linear predicates.

Theorem 4.2 *Any linear predicate is also semi-linear.* **Proof:** *Let ϕ be any linear predicate. To show that it is semi-linear, consider any X such that $\neg\phi(X) \wedge \neg final(X)$. From linearity of ϕ there exists a state $s \in sup(X)$ such that s is forbidden in X. We will show that there exists a state t in X which is eligible and semi-forbidden. If the state s itself is eligible, then we are done since $forb_\phi(s, X)$ clearly implies $sforb_\phi(s, X)$. Otherwise, let t be an eligible state in X such that $t \rightarrow succ(s)$. Since $\neg final(X)$ we are guaranteed existence of such a t. We only need to show that t is semi-forbidden in X. Consider any Y such that $X^t \leq Y$. If $\neg(X^s \leq Y)$, then from linearity we get that $\neg P_\phi(X, Y)$. Thus, t is semi-forbidden in this case. Now, let Y be such that $X^s \leq Y$ and $P_\phi(X, Y)$. Since $\neg\phi(X)$, there exists W such that $X < W \leq Y$ and $\phi(W)$. From linearity of ϕ, it follows that $X^s \leq W$. This implies $X^t \leq W$ since $t \rightarrow succ(s)$. Therefore, $P_\phi(X, Y) \Rightarrow P_\phi(X^t, Y)$ and thus t is semi-forbidden.* 2

We now show that semi-linear includes the class observer-independent.

Theorem 4.3 *Any observer-independent predicate is also semi-linear.* **Proof:** *Let ϕ be an observer-independent predicate and X be any cut such that $\neg\phi(X)$ and $\neg final(X)$. We show that for any state $s \in eligible(X)$, s is semi-forbidden. Consider any Y such that $X^s \leq Y$. If $\neg P_\phi(X, Y)$, s is semi-forbidden trivially. Otherwise, assume that $P_\phi(X, Y)$ holds. Since ϕ is observer independent, it follows that $D_\phi(X, Y)$. If $\phi(X^s)$ then s is semi-forbidden since $P_\phi(X^s, Y)$ is trivially true.*

Alternatively, assume ϕ is not true in X^s. Since $\neg\phi(X)$, we know $\neg P_\phi(X, X^s)$. Since $D_\phi(X, Y)$, we can conclude $\exists W : X^s < W \leq Y : \phi(W)$. Again, s is semi-forbidden. 2

Note that the observer-independent and semi-linear classes are not equivalent. As shown above, a conjunctive predicate is linear (and hence also semi-linear), but not observer-independent.

4.4.1 Examples of semi-linear predicates

For an example of a semi-linear predicate, consider the execution of a mutual exclusion algorithm. Let $X[i]$ denote the supremal state from process p_i in X. Let $CS(X[i])$ denote that process p_i is in the critical section in state $X[i]$. To ensure that the given execution is proper, we are interested in determining existence of a consistent cut X such that $\phi(X) \stackrel{\text{def}}{=} \exists i, j : i \neq j : CS(X[i]) \wedge CS(X[j])$.

Theorem 4.4 $\phi(X) \stackrel{\text{def}}{=} \exists i, j : CS(X[i]) \wedge CS(X[j])$ *is semi-linear.* **Proof:** *Assume that* $\neg\phi(X)$. *Therefore,*

$$\forall i, j : \neg CS(X[i]) \vee \neg CS(X[j])$$

If X *is the final global state, then we are done. Otherwise, eligible*(X) *is non-empty. We now do a case analysis.*
Case 1*: There exists* $s \in$ *eligible*(X) *such that* $\neg CS(s)$.
We claim that s *satisfies sforb*$_\phi(s, X)$. *Let* Y *be any extension such that* $X^s \leq Y$. *If* $\neg P_\phi(X, Y)$, *then we are done. Else, let there be* W *such that* $X \leq W \leq Y$ *and* $\phi(W)$. *If* $X^s \leq W$, *then* s *is semi-forbidden. From Lemma 4.1 part 1,* W^s *is a consistent cut. It is sufficient to show that* W^s *satisfies* ϕ. *From* $\phi(W)$, *there exist* i, j *such that* $CS(W[i]) \wedge CS(W[j])$. *Since we have assumed in this case* $\neg CS(s)$, $s \neq W[i] \wedge s \neq W[j]$. *Hence,* $W[i] = W^s[i] \wedge W[j] = W^s[j]$. *Thus* $\phi(W^s)$, *and* s *is semi-forbidden in* X.
Case 2*: There does not exist* s *in eligible*(X) *such that* $\neg CS(s)$.
Since eligible(X) *is non-empty, it follows that there is a unique* s *in eligible*(X) *such that* $CS(s)$. *We again claim that* s *satisfies sforb*$_\phi(s, X)$. *Let* Y *and* W *be as in case 1. Since* $X < W$, *and eligible*(X) *consists of singleton* $\{s\}$, *it follows from Lemma 4.1 part 2 that* $X^s \leq W$. *Therefore,* s *is semi-forbidden.* 2

As another example of a semi-linear predicate consider a disjunction of local predicates. Since this predicate is observer-independent, it follows from Lemma 4.3 that it is also semi-linear.

4.5 Closure Properties

4.5.1 Stable is closed under conjunction and disjunction

It is easy to see that the conjunction of two stable predicates is also stable. Let $\phi \equiv \phi_1 \wedge \phi_2$ where ϕ_1 and ϕ_2 are both stable predicates. Consider a pair of cuts, X and Y where $\phi(X)$ is true and $X \leq Y$. Note that $\phi_1(X)$ is true, and hence so is $\phi_1(Y)$ (because of the stability of ϕ_1). Similarly $\phi_2(Y)$ is true. Hence, $\phi(Y)$ is true. A

similar argument can be used to show that the class of stable predicates is closed under disjunction.

4.5.2 Observer Independent is closed under disjunction

A proof that the class of observer independent predicates is closed under disjunction was first given in [5]. Consider $\phi \equiv \phi_1 \vee \phi_2$, where both ϕ_1 and ϕ_2 are observer independent. Let X and Y be cuts such that $P_\phi(X, Y)$. By the definition of $P_\phi(X, Y)$, there must exist some cut $W : X \leq W \leq Y$ where $\phi(W)$. Without loss of generality, assume $\phi_1(W)$, and hence $P_{\phi_1}(X, Y)$. Since ϕ_1 is observer independent, we know that $D_{\phi_1}(X, Y)$. Therefore $D_\phi(X, Y)$ and ϕ is observer independent.

However, the conjunction of two observer independent predicates may not be observer independent. A simple counter example is $\phi = a \wedge b$, for boolean variables a and b on different processes (see Section 4.3). However, the predicate "a is true" is observer-independent, since a is local to a single process (similarly for b).

4.5.3 Linear is closed under conjunction

The class of linear predicates is closed under conjunction. Let ϕ_1 and ϕ_2 be any linear predicates. Denote by $\text{forb}_{\phi_i}(s, X)$ that state s is forbidden with respect to ϕ_i in cut X. Let ϕ be the conjunction of ϕ_1 and ϕ_2. Then we show:

$$[\text{forb}_{\phi_1}(s, X) \vee \text{forb}_{\phi_2}(s, X)] \Rightarrow \text{forb}_\phi(s, X)$$

Assume $\text{forb}_{\phi_1}(s, X)$, and let Y be reachable from X. If $X^s \not\leq Y$, then we know by the definition of forbidden that $\neg\phi_1(Y)$. Hence $\neg\phi(Y)$. Similarly, $\text{forb}_{\phi_2}(s, X)$ implies $\text{forb}_\phi(s, X)$.

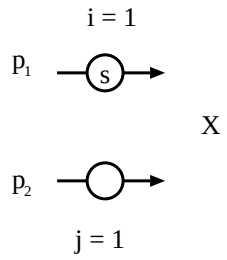
Assume $\neg\phi(Y)$. We now show that there exists a forbidden supremal state in Y. Without loss of generality, assume $\neg\phi_1(Y)$. Since ϕ_1 is linear, there must exist a supremal state, s, in Y such that $\text{forb}_{\phi_1}(s, Y)$. As shown above, we know $\text{forb}_\phi(s, Y)$.

4.5.4 Linear is not closed under disjunction

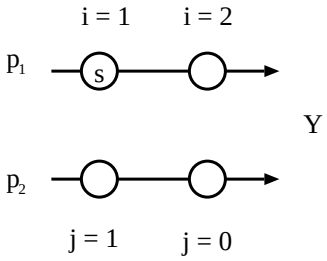
The class of linear predicates is not closed under disjunction. A disjunction of local predicates, such as "$\phi \stackrel{\text{def}}{=} a \vee b$", is a simple counter example. Consider a cut X from a system with two processes. Assume $\neg\phi(X)$. Let s be the supremal state in X for variable a, and t be the supremal state in X for variable b. We show that s is not forbidden. If b is true in succ(t), then ϕ will be true in X^t. Similarly, t is not forbidden. Thus, there are no states in X that are forbidden, and hence ϕ is not linear.

4.5.5 Semi-linear is not closed under conjunction or disjunction

The class of semi-linear predicates is not closed under either conjunction or disjunction. A single counter example of a predicate that is not semi-linear can serve to prove both the conjunctive and disjunctive cases. First, we explain the example. Consider a predicate defined on integer values, i and j, from two processes. The predicate $\phi \stackrel{\text{def}}{=} i + j < 2$ is not semi-linear. The following execution shows a cut in which neither of the supremal states are semi-forbidden.



Without loss of generality, assume the state, s, is semi-forbidden. Then consider the following extension:



Note that, $\neg\phi(X)$, $X^s \leq Y$, and $P_\phi(X, Y)$. However, the predicate is only true in Y when i takes the value 1 and j takes the value 0. No cut W exists where $X^s \leq W \leq Y$ and $\phi(W)$. Thus s is not semi-forbidden. By symmetry, neither of the states in X are semi-forbidden. Therefore ϕ is not semi-linear.

We now rewrite $i + j < 2$ as an equation of boolean variables, a, b, c and d. Let i be encoded in two bits, a and b, and j be encoded by bits c and d. Then we have:

$$\phi \stackrel{\text{def}}{=} (\neg a \wedge \neg c) \wedge (\neg b \vee \neg d)$$

This predicate is a conjunction of two terms. The first term is itself a conjunctive predicate ($\neg a \wedge \neg c$). The second term is a disjunctive predicate ($\neg b \vee \neg d$). Both terms are semi-linear predicates, yet their conjunction is not semi-linear.

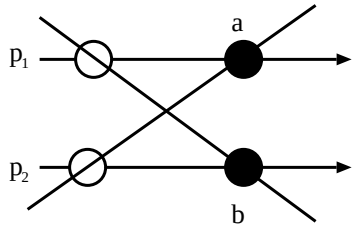
Also, by rewriting ϕ in disjunctive normal form as:

$$\phi \stackrel{\text{def}}{=} (\neg a \wedge \neg c \wedge \neg b) \vee (\neg a \wedge \neg c \wedge \neg d)$$

we have a disjunction of two conjunctive predicates. Thus we have two semi-linear predicates ($\neg a \wedge \neg c \wedge \neg b$ and $\neg a \wedge \neg c \wedge \neg d$), whose disjunction is not semi-linear.

4.6 Linear Predicates have an Infimum Satisfying Cut

In some applications, most notably distributed debugging, it is important to detect the first cut in which a predicate becomes true. For example, a programmer may be attempting to determine what programming error causes the system to enter an undesirable state. If the debugger can identify the first such state, the programmer can more easily trace the problem to its source. However, for some predicates there is no unique first cut in which the predicate is true. That is, although the set of consistent cuts is a lattice, the set of all cuts for which ϕ is true, may not be a lattice. For example, consider a disjunctive predicate, $\phi \stackrel{\text{def}}{=} a \vee b$. In the following diagram there are two infimal cuts (shown with solid lines) for which the predicate is true. There is no unique first cut for which ϕ is true.



Denote by $\mathcal{C}_\phi$ the set of *all* cuts for which a predicate ϕ is true, and by $\mathcal{C}_\phi(X, Y)$ the set of cuts between X and Y for which ϕ is true. We now show that for any X, Y, the set $\mathcal{C}_\phi(X, Y)$ contains a unique infimum if and only if ϕ is linear.

Lemma 4.5 *$\mathcal{C}_\phi$ is an inf-semilattice iff ϕ is linear.* **Proof:** *($\mathcal{C}_\phi$ is an inf-semilattice $\Rightarrow$ ϕ is linear)*
We prove the contrapositive. Assume that ϕ is not linear. From the definition of linearity, there exists some cut X such that none of the supremal states in X are forbidden. That is, for any supremal state s in X:

$$\exists Y_s : X \leq Y_s \wedge X^s \not\leq Y_s : \phi(Y_s)$$

Note that X is equal to the intersection of all Y_s. Each cut Y_s is an element of $\mathcal{C}_\phi$, but their infimum (X) is not in $\mathcal{C}_\phi$. Therefore, $\mathcal{C}_\phi$ is not an inf-semilattice.

(ϕ is linear $\Rightarrow$ $\mathcal{C}_\phi$ is an inf-semilattice)
We again show the contrapositive. Let $C = \{Y_1, Y_2, ..Y_k\}$ be any subset of $\mathcal{C}_\phi$ such that inf $C \notin \mathcal{C}_\phi$. We know that inf C must exist, since the set of consistent cuts forms a lattice. Let $X = inf\ C$.

Given any supremal state $s \in X$, there exists $Y_i \in C$ such that s is a supremal state in Y_i. Note that $X \leq Y_i$, but $X^s \not\leq Y_i$. Furthermore, $\phi(Y_i)$ while $\neg\phi(X)$. Thus, s is not forbidden in X. Since there are no forbidden supremal states in X, ϕ is not linear. 2

4.7 Dual Properties

Just as existence of the least cut requires that the predicate ϕ be linear, the existence of the *largest* satisfying cut requires a property that is dual of linearity.

Definition 4.6 *A predicate ϕ is post-linear iff*

$$\forall X : \neg\phi(X) \Rightarrow \exists s \in sup(X) : \forall Y \leq X : Y^s \leq X \vee \neg\phi(Y)$$

For example, for $\phi \stackrel{\text{def}}{=} b + c < k$, if b is known to be monotonically decreasing, then the predicate is post-linear.

All the results for linear predicates have dual versions for post-linear predicates. Thus, ϕ is a post-linear predicate iff $\mathcal{C}_\phi$ is a sup-semilattice. Combining these results, we get:

Theorem 4.7 *$\mathcal{C}_\phi$ is a lattice iff ϕ is linear and post-linear.*

As an application of Theorem 4.7, we consider the problem of recovery in distributed systems [18]. We call a local state *recoverable* if after a failure, the state can be recovered from the disk using a checkpoint and the message log. A cut is called recoverable if all states belonging to that cut are recoverable and the cut is consistent [3]

The following is an easy corollary of the Theorem 4.7.

Corollary 4.8 *The set of all recoverable cuts is a lattice.* **Proof:** *Recoverability of a state is local to a process. Any local predicate is both linear and post-linear.* 2

Since results for dual properties are easily derived, we will not discuss them any further.

[3] Note that the notion of consistency in [18] is slightly different from the one discussed in this paper.

Algorithm A_{general}
Given execution $[X, Y]$ and predicate ϕ
(0) choose W between X and Y
(1) **while** $\neg\phi(W)$ and $W < Y$ **do**
(2) advance W;
(3) **end while;**
(4) return $\phi(W)$;

Figure 2: Algorithm to detect a global predicate

5 Detection

We now discuss predicate detection for different classes of global predicates. Note that, detection of a predicate is specific to a single execution. That is, predicate detection is not a verification process that asserts the predicate becomes true (or remains false) in all *possible* executions. Only that it became true (remained false) for one particular execution.

We therefore treat the detection problem as one in which a partial execution from cut X to Y is provided as input to an algorithm A. The algorithm responds with the value for $P_\phi(X, Y)$. If the algorithm evaluates to true, then the algorithm also returns a cut W for which $\phi(W)$.

All the algorithms use a common paradigm shown in Fig. 2. We refer to this algorithm as A_{general}. Algorithm A_{general} repeatedly checks the value of the predicate on some cut W. If the predicate is false and W is not the last cut in the computation, we advance W by setting it to some cut that is reachable from the old value of W. Note we require that $W \leq Y$ be maintained when W is advanced.[4] This algorithm is non-deterministic in two ways. It does not specify the initial value of W, and it does not specify how W should be advanced. We will see later that by restricting this non-determinism, algorithms for various classes of predicates can be derived.

Any execution of the detection algorithm corresponds to a sequence of cuts (the successive values of W). Let $S = \{S.1, \ldots S.\text{last}\}$ denote one such sequence. That is, the $S.i$ are successive values of variable W that are evaluated at line (1). Let sequence(S, X, Y, ϕ, A) denote the property that S is a possible sequence of cuts produced by algorithm A during detection of the global predicate ϕ in execution $[X, Y]$. For the above algorithm, an alternative characterization of sequence$(S, X, Y, \phi, A_{\text{general}})$ is given below.

[4] We also assume that the predicate final(s) is true for all supremal states s in Y.

Lemma 5.1 *sequence*$(S, X, Y, \phi, A_{general}) \stackrel{\text{def}}{=}$

$$\forall i : S.i < S.(i+1) \wedge$$
$$\phi(S.last) \vee Y \leq S.last \wedge$$
$$\forall i : i < last : \neg\phi(S.i) \wedge$$
$$\forall i :: X \leq S.i \leq Y$$

Proof: *The first conjunct follows because successive $S.i$'s are obtained by advancing W. The second conjunct is obtained by negating the guard on the* while *loop. The third conjunct specifies the condition under which the evaluation of ϕ is done on an additional cut. The final conjunct follows from the initialization of W, and the definition of "advance".*

Conversely, any sequence S that satisfies these four conjuncts is a possible sequence of the algorithm. 2

What does it mean to say that algorithm A detects the predicate ϕ for execution Y? Since the algorithm returns $\phi(S.\text{last})$ as its value, the algorithm is correct only if $P_\phi(X, Y)$ is equal to $\phi(S.\text{last})$. Since the detection algorithm is non-deterministic, it must return this answer for any sequence S that satisfies sequence(S, X, Y, ϕ, A). Thus, we can formally define the property that algorithm A detects a predicate ϕ as

$$\text{detect}(A, \phi) \stackrel{\text{def}}{=} \forall X, Y, S : sequence(S, X, Y, \phi, A) : P_\phi(X, Y) = \phi(S.\text{last})$$

Since $\phi(S.\text{last})$ always implies $P_\phi(X, Y)$ we can rewrite *detect* as a:

$$\text{detect}(A, \phi) = \forall X, Y, S : sequence(S, X, Y, \phi, A) : P_\phi(X, Y) \Rightarrow \phi(S.\text{last})$$

We now discuss how various detection algorithms can be derived from Algorithm A_{general}.

5.1 Stable Predicates

In this section we show that if we do not constrain any non-determinism, the general algorithm detects a predicate ϕ if it is stable. The converse is also true. If a predicate is not stable, the general algorithm cannot correctly detect it for some execution. Therefore, $A_{\text{stable}} = A_{\text{general}}$.

Theorem 5.2 *For any predicate ϕ,*

$$\mathit{detect}(A_{\mathit{stable}}, \phi) \equiv \mathit{stable}(\phi)$$

Proof: *First assume that ϕ is stable. From the definition of* detect *we need to show that*

$$\forall X, Y, S : \mathit{sequence}(S, X, Y, \phi, A_{\mathit{stable}}) : P_\phi(X, Y) \Rightarrow \phi(S.last)$$

For any X, Y, S assume that sequence$(S, X, Y, \phi, A_{\mathit{stable}})$ and $P_\phi(X, Y)$ hold. From Lemma 5.1 conjunct 2, it follows that $\phi(S.last) \vee S.last = Y$. The first disjunct gives us the desired result, so now consider $S.last = Y$. However, $P_\phi(X, Y)$ and stable(ϕ) imply $\phi(Y)$. Combining this with $S.last = Y$ we get $\phi(S.last)$.

For the converse, assume that ϕ is not stable. This implies that there exists $[X, Y]$ such that $P_\phi(X, Y)$ and $\neg\phi(Y)$. Let the sequence S be simply defined as a sequence consisting of a single cut — Y. It is easy to verify that sequence$(S, X, Y, \phi, A_{\mathit{stable}})$ and $P_\phi(X, Y)$ hold but $\phi(S.last)$ is false. Thus, we get detect$(A_{\mathit{stable}}, \phi)$ is false. 2

5.2 Observer Independent Predicates

For the general algorithm to work correctly for observer independent predicates, we need to restrict its non-determinism. First, we require that the algorithm begin its search from X. Second, we require that while advancing the cut W, the algorithm advances by exactly one eligible local state from W.[5] Observe that the resulting algorithm, $A_{\text{o-i}}$, shown in Fig. 3, is still non-deterministic, since it can choose any eligible state to advance the cut. There may be multiple eligible states in any cut.

We now give a characterization of sequence$(S, X, Y, \phi, A_{\text{o-i}})$. First, a path, S, in the execution $[X, Y]$ is defined as:

$$\text{path}(S, X, Y) \stackrel{\text{def}}{=} (S.1 = X) \wedge \forall i :: (\exists s \in \text{eligible}(S.i) : S.(i+1) = S.i^s)$$

That is, each $S.i$ and $S.(i+1)$ differ by exactly one eligible state. From the algorithm, it then follows that sequence$(S, X, Y, \phi, A_{\text{o-i}}) =$ (sequence$(S, X, Y, \phi, A_{\text{stable}}) \wedge$ path(S, X, Y)).

We now show that the algorithm captures the class of observer-independent predicates.

[5] The theory can be easily generalized to handle the case of simultaneously advancing along multiple eligible states. For simplicity of discussion, we have restricted this paper to a single eligible state.

Algorithm $A_{\text{o-i}}$
Given execution $[X, Y]$ and predicate ϕ
(0) $W := X$;
(1) **while** $\neg\phi(W)$ and $W < Y$ do
(2) advance W by any eligible state;
(3) **end while**;
(4) **return** $\phi(W)$;

Figure 3: Algorithm to detect a observer independent predicate

Theorem 5.3 *For any predicate ϕ,*

$$\textit{detect}(A_{\textit{o-i}}, \phi) \equiv \textit{observer-independent}(\phi)$$

Proof: *First assume that ϕ is observer independent. From the definition of $detect$, we need to show that*

$$\forall X, Y, S : \textit{sequence}(S, X, Y, \phi, A_{\textit{o-i}}) : P_\phi(X, Y) \Rightarrow \phi(S.\textit{last})$$

For any X, Y, S assume that $\textit{sequence}(S, X, Y, \phi, A_{\textit{o-i}})$ and $P_\phi(X, Y)$ hold. From $\textit{sequence}(S, X, Y, \phi, A_{\textit{o-i}})$ either $\phi(S.\textit{last})$ or $S.\textit{last} = Y$. In the former case, we are done. So now assume that $S.\textit{last} = Y$. From the definition of observer-independent(ϕ), we obtain that $D_\phi(X, Y)$ holds. From $\textit{path}(S, X, Y)$, $(S.last = Y)$ and $D_\phi(X, Y)$, it follows that $\exists i : \phi(S.i)$. Combining this with the fact that $\forall i : i < \textit{last} : \neg\phi(S.i)$, we get that $\phi(S.\textit{last})$ holds.

Now assume that ϕ is not observer-independent. This implies that there exists $[X, Y]$ for which $P_\phi(X, Y)$ is true but $D_\phi(X, Y)$ is false. From the latter, there exists a sequence S such that $\textit{path}(S, X, Y)$, $(S.\textit{last} = Y)$ and $\forall i : \neg\phi(S.i)$. It is easy to see that $\textit{sequence}(S, X, Y, \phi, A_{\textit{o-i}})$ and $P_\phi(X, Y)$ hold; however, $\phi(S.\textit{last})$ is false. Therefore, $\textit{detect}(A_{\textit{o-i}}, \phi)$ is false. 2

Observe that any sequence of the algorithm $A_{\text{o-i}}$ is also a sequence of the algorithm A_{stable}. This is an alternative proof that the class of stable predicates is contained in the class of observer-independent predicates.

5.3 Semi-linear Predicates

For semi-linear predicates, we need to restrict the non-determinism further. Instead of advancing the cut W using *any* eligible state, we will advance it using the state determined by a function f. That is, we are given some function f whose domain

Algorithm A_f
Given execution $[X, Y]$, semi-linear ϕ, and function f
(0) W := X;
(1) **while** $\neg\phi(W)$ and $W < Y$ do
(2) advance W by $f(W)$;
(3) **end while**;
(4) **return** ϕ(W);

Figure 4: Algorithm to detect a semi-linear predicate given semi-forbidden function f

is the set of cuts. When the function is evaluated, it returns one of the eligible states in the cut. The function may use the knowledge about the predicate and the cut to determine how the cut should be advanced. We will show that for detecting semi-linear predicates, the function f must evaluate to a semi-forbidden state.

The resulting algorithm, A_f is shown in Fig. 4. It is easy to see that:

$$\text{sequence}(S, X, Y, \phi, A_f) = [\text{sequence}(S, X, Y, \phi, A_{\text{o-i}}) \wedge (\forall i : S.(i{+}1) - S.i = f(S.i))]$$

Theorem 5.4 *For any predicate ϕ:*

$$(\exists f : \textit{detect}(A_f, \phi)) \equiv \textit{semi-linear}(\phi)$$

Proof: *First assume that ϕ is semi-linear. From definition of $detect$, we need to show that there exists f such that*

$$\forall X, Y, S : \textit{sequence}(S, X, Y, \phi, A_f) : P_\phi(X, Y) \Rightarrow \phi(S.\textit{last})$$

Let f be any function that evaluates to a semi-forbidden, supremal state in any cut W for which $\neg\phi(W)$. Such a function must exist by the definition of semi-linear ϕ.

For any X, Y, S assume that sequence(S, X, Y, ϕ, A_f) and $P_\phi(X, Y)$ hold. As before, it is sufficient to consider the case when $S.last = Y$.

$P_\phi(X, Y)$ implies $\exists W : X \leq W \leq Y : \phi(W)$. If $W = Y$, then we are done. Otherwise, let W be an infimal cut in $\mathcal{C}_\phi$ and let $S.i$ be such that such that $S.i < W$. Note that, $\neg\phi(S.i)$. By the definition of the function f, there exists s such that $(S.i)^s = S.(i+1)$ and sforb$_\phi(s, S.i)$.

By the definition of semi-forbidden, we know $P_\phi(S.(i+1), Y)$. That is, there must exist a cut $W' \leq Y$ such that $S.(i+1) \leq W'$ and $\phi(W')$. If $S.(i+1) = Y$, then we are done. Otherwise repeat this analysis using W' and $S.(i+1)$ instead of W and $S.i$. Since S is finite, we must eventually arrive at $S.last$.

Now for the converse, assume that there exists some f such that algorithm A_f detects ϕ, and that ϕ is not semi-linear. We show an adversary who will force the algorithm to make a mistake.

Non-semilinearity of ϕ implies that there exists some execution X, such that $\neg\phi(X)$ and none of the $s \in$ eligible(X) are semi-forbidden. That is, for each s in eligible(X) there exists an extension $Y_s : X^s \leq Y_s : P_\phi(X, Y_s)$ but not $P_\phi(X^s, Y_s)$.

The adversary asks the algorithm to begin execution with cut X. Let s be the result of $f(X)$. The adversary presents the algorithm with a successor of s, allowing the algorithm to advance to X^s. At this point, the adversary provides to the algorithm only the remaining execution of Y_s. Note that the algorithm cannot halt in a state W for which $\phi(W)$ is true. Yet, $P_\phi(X, Y_s)$ is true. Thus the algorithm does not detect ϕ in this case. 2

Since the class of linear predicates is contained within the semi-linear class, algorithm A_f can also be used to detect linear predicates. Note, however, that if f evaluates to a forbidden state, then A_f will not only detect ϕ, but it will also return the first cut for which ϕ is true.

Theorem 5.5 $(\forall X : f(X) = s \Rightarrow$ *forb*$_\phi(s, X)) \wedge$ *sequence*$(S, X, Y, \phi, A_f) \wedge \phi(S.last)$ *implies that* $S.last =$ *inf* $\mathcal{C}_\phi(X, Y)$. **Proof:** *The proof is by contradiction. Assume* $S.last \neq$ *inf* $\mathcal{C}_\phi(X, Y)$. *Let W denote the infimum of $\mathcal{C}_\phi(X, Y)$. Clearly, $W \neq X$. Let i be the largest sequence index such that $S.i \leq W$. From the algorithm, this implies that there exists a process index k such that $W[k] = f(S.i)$. This implies* forb$_\phi(W[k], S.i)$ *and since we know $S.i \leq W$, we can conclude $\neg\phi(W)$, contradicting our assumption.* 2

6 Conclusions

We have shown that the general problem of detecting a global predicate is NP-complete. We have defined two classes of predicates — linear and semi-linear. We show that linear predicates are exactly those for which the first satisfying cut is unique. We show that the semi-linear class contains the linear class, as well as the previously known classes of stable predicates and observer-independent predicates. We demonstrate the closure properties of each of these classes with respect to conjunction and disjunction.

We also provide a family of algorithms for detecting the occurrence of a predicate within a range of execution. We show that the algorithms are exact in the sense that they detect precisely those predicates within the respective class. The most non-deterministic algorithm will detect exactly the stable class of predicates.

By successively constraining the non-determinism of this algorithm, we derive algorithms for the observer-independent and semi-linear classes. All of the algorithms are efficient and can be used for on-line monitoring.

Acknowledgments

We gratefully acknowledge many valuable discussions with Alex Tomlinson concerning predicate linearity.

References

[1] O. Babaoğlu and K. Marzullo. Consistent global states of distributed systems: Fundamental concepts and mechanisms. In Sape Mullender, editor, *Distributed Systems*, pages 55–96. Addison Wesley, New York, NY, 2nd edition, 1994.

[2] Ö. Babaoğlu and M. Raynal. Specification and detection of behavioral patterns in distributed computations. In *Proc. of 4th IFIP WG 10.4 Int. Conference on Dependable Computing for Critical Applications*, San Diego, CA, January 1994. Springer Verlag Series in Dependable Computing.

[3] L. Bouge. Repeated snapshots in distributed systems with synchronous communication and their implementation in CSP. *Theoretical Computer Science*, 49:145–169, 1987.

[4] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[5] Bernadette Charron-Bost, Carole Delporte-Gallet, and Hugues Fauconnier. Local and temporal predicates in distributed systems. *ACM Transactions on Programming Languages and Systems*, 17(1):157–179, jan 1995.

[6] Craig M. Chase and Vijay K. Garg. Efficient detection of restricted classes of global predicates. In *Lecture Notes in Computer Science: Proceedings of the 9th International Workshop on Distributed Algorithms*, pages 303–317. Springer Verlag, 1995.

[7] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. of the Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, CA, May 1991. ACM/ONR.

[8] R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, California, May 1991.

[9] E. Fromentin, M. Raynal, V. K. Garg, and A. I. Tomlinson. On the fly testing of regular patterns in distributed computations. In *Proc. of the 23rd Intl. Conf. on Parallel Processing*, pages 2:73 – 76, St. Charles, IL, August 1994.

[10] V. K. Garg. Observation of global properties in distributed systems. In *IEEE Internation Conference on Software and Knowledge Engineering*, Lake Tahoe, NV, jun 1996. to appear.

[11] V. K. Garg, C. Chase, R. Kilgore, and J. R. Mitchell. Detecting conjunctive channel predicates in a distributed programming environment. In *Proc. of the International Conference on System Sciences*, volume 2, pages 232–241, Maui, Hawaii, January 1995.

[12] V. K. Garg and B. Waldecker. Detection of unstable predicates in distributed programs. In *Proc. of 12th Conference on the Foundations of Software Technology & Theoretical Computer Science*, pages 253–264. Springer Verlag, December 1992. Lecture Notes in Computer Science 652.

[13] V. K. Garg and B. Waldecker. Detection of weak unstable predicates in distributed programs. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):299–307, March 1994.

[14] Vijay. K. Garg, Craig. M. Chase, J. R. Mitchell, and R. Kilgore. Detecting conjunctive channel predicates in a distributed programming environment. In *Proc. of the International Conference on System Sciences*, volume 2, pages 232–241, Maui, Hawaii, January 1995.

[15] Bojan Groselj. Bounded and minimum global snapshots. *IEEE Parallel and Distributed Technology*, 1(4):72–83, 1993.

[16] M. Hurfin, N. Plouzeau, and M. Raynal. Detecting atomic sequences of predicates in distributed computations. In *Proc. of the Workshop on Parallel and Distributed Debugging*, pages 32–42, San Diego, CA, May 1993. ACM/ONR. (Reprinted in SIGPLAN Notices, Dec. 1993).

[17] R. Jégou, R. Medina, and L. Nourine. Linear space algorithm for on-line detection of global predicates. In *Proceedings of the International Workshop on Structures in Concurrency Theory (STRICT '95)*, pages 175–189, Berlin, Germany, 1995.

[18] D. B. Johnson and W. Zwaenepoel. Recovery in distributed systems using optimistic message logging and checkpointing. *Journal of Algorithms*, 11(3):462–491, September 1990.

[19] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[20] Keith Marzullo and Gil Neiger. Detection of global state predicates. In *Proceedings of the 5th International Workshop on Distributed Algorithms*, pages 254–272, 1991.

[21] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms: Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226. Elsevier Science Publishers B. V, 1989.

[22] B. P. Miller and J. Choi. Breakpoints and halting in distributed programs. In *Proc. of the* 8^{th} *International Conference on Distributed Computing Systems*, pages 316–323, San Jose, CA, July 1988. IEEE.

[23] R. Schwartz and F. Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[24] M. Spezialetti and P. Kearns. Efficient distributed snapshots. In *Proc. of the* 6^{th} *International Conference on Distributed Computing Systems*, pages 382–388, 1986.

[25] Scott D. Stoller and Fred B. Schneider. Faster possibility detection by combining two approaches. In *Lecture Notes in Computer Science: Proceedings of the 9th International Workshop on Distributed Algorithms*, pages 318–332. Springer Verlag, 1995.

[26] A. I. Tomlinson and V. K. Garg. Detecting relational global predicates in distributed systems. In *Proc. of the Workshop on Parallel and Distributed Debugging*, pages 21–31, San Diego, CA, May 1993. ACM/ONR.