On Reducing the Global State Graph for Verification of Distributed Computations

Arindam Chakraborty and Vijay K. Garg
Parallel and Distributed Systems Laboratory
The University of Texas at Austin
Austin, TX 78712, USA
{chakrabo,garg}@ece.utexas.edu

Abstract

Correct distributed programs are very hard to write and reason about. Verification of distributed programs with respect to their specifications is thus very important to ensure that a distributed system works as expected. Model checking has emerged as a technique used to verify the correctness of programs. However model checking techniques suffer from the global state explosion problem - finite state models of distributed and concurrent systems grow exponentially in size as the number of components in the system increases. Even though work has been done to reduce the size of the global state graph generated by model checking algorithms, current research has mainly focussed on finite-state transition graphs. In the context of distributed computations, additional advantage can be derived from the fact that the global state graph forms a distributive lattice. We present a generic technique using congruences to reduce the global state graph of distributed programs, by using lattice theoretic properties of the graph. Our state space reduction technique is generic in the sense that it integrates seamlessly with any model checking algorithm.

1 Introduction

Writing distributed programs is an error prone activity. Not surprisingly, distributed systems are particularly vulnerable to software faults. It is hard to reason about them because they suffer from the combinatorial explosion problem. Verification and validation of distributed programs and software fault-tolerance is an important way to ensure reliability of distributed systems. Detecting a fault in an execution of a distributed system is a fundamental problem that arises during the verification process. For example, when testing a distributed mutual exclusion algorithm, it is useful to monitor safety properties such as, "there are no concurrent accesses to the critical sections", as well as liveness or progress properties such as, "once a process makes a request to access the critical section, it is eventually granted access".

In an *asynchronous* distributed system due to unsynchronized clocks and lack of bounds on processor speed and network latency, the order in which events on different processes actually occurred cannot in general be determined. Therefore the sequence of global states through which the system has passed cannot be uniquely determined. Thus verifying whether a property held in an execution of a distributed system is difficult. Lamport [12] introduced the *happened-before* relation, a partial order that reflects causal dependencies between events. A history of an asynchronous distributed system can be approximated by a *computation*, which comprises of the local computation of each process together with the happened-before relation. This is useful for verification as the happened-before relation can be determined by using vector clocks [9, 13]. Since this is a partial order, the history is not uniquely determined. Instead it restricts the possibilities to all histories that are *consistent* with the computation C (all total orders on the events in C that contain the happened-before relation). A *consistent global state* (CGS) of a computation C is a global state that appears in some history consistent with C. The set of all consistent global states forms an algebraic structure called a *distributive lattice*.

Cooper and Marzullo [7] first addressed the problem of verifying distributed computations by formulating the problem of *predicate detection* and introducing two modalities of detecting predicates - *possibly* and *definitely*. A predicate Φ is said to hold *possibly* over a computation C iff, in some history consistent with C, the system passes through a global state that satisfies Φ. A predicate is said to hold *definitely* over a computation iff, in all histories consistent with C, the system passes through a global state that satisfies Φ. Cooper and Marzullo give centralized algorithms for detecting $possibly : \Phi$ and $definitely : \Phi$ based on breadth first search of the global state space (or lattice). Each process reports its local states to a central monitor, which incrementally constructs a lattice of all the CGSs of the computation.

However, the global state space generated by a computation suffers from the global state explosion problem. In general, the number of global states is exponential in the number of system components. In a distributed system of n processes with a maximum of m events per process, the total number of possible global states is of $O(m^n)$. The approach of Cooper and Marzullo suffers from this problem. This has motivated the development of efficient algorithms for detecting restricted classes of predicates [11]. In this paper, we focus on verifying properties of distributed computations based on Cooper and Marzullo's approach. This is because (1) the existing polynomial time algorithms are for restricted forms of predicates, and (2) the polynomial time algorithms are different for different classes of predicates.

Alagar and Venkatesan [1] have explored the question of alleviating this problem of global state explosion problem for detecting $possibly : \Phi$ for any arbitrary predicate. They improve the performance of the algorithms by increasing the granularity of execution step from an event to a sequence of events (*interval*). Instead of testing every global state, *global intervals* are tested. When the values of the variables related to the global predicates are not changed frequently, the number of global intervals can be substantially less than the number of global states, thereby reducing the space and time complexity of global space search algorithms. Similar ideas have been explored by Marzullo and Neiger [14] based on *weak vector clocks*. It can be shown that the equivalence relation on the global state space generated by weak vector clocks is finer than the equivalence relation formed by interval clocks. Thus interval clocks lead to greater state space reduction.

The approach of Cooper and Marzullo restricts the expressiveness of properties to the *possibly* and *definitely* modalities. Alagar and Venkatesan addressed the issue of global state explosion, but restrict the expressiveness of properties to only the *possibility* modality. Traditionally system properties have been expressed using *temporal* predicates. Temporal predicates are more powerful and expressive but are also more complex to use. The *model checking* [4, 6] problem is to decide whether a finite-state description of a reactive system satisfies a temporal-logic specification. The model-theoretic approach mechanically determines if the system meets a specification expressed in propositional temporal logic. The global state graph of the concurrent system is constructed and then the model checking algorithm is used to determine whether the program meets its specification. Model checking algorithms are similar to global flow analysis algorithms and have complexity linear in the size of the structure and the specification. Thus even though the expressibility of system properties is increased, the problem of global state space explosion remains.

We apply model checking techniques to distributed computations by considering the distributive lattice of the computation to be the global state graph. Since the distributive lattice structure satisfies special properties, these can be exploited to reduce the global state lattice to verify properties, without restricting the expressiveness of these properties to special modalities or types of predicates. We show that the concept of *global intervals* is a special case of merging states on the distributive lattice corresponding to the computation called *lattice congruences*. We can reduce the number of states in the global state space lattice on the basis of similarities between different states. With respect to a given property Φ, a set of global states may be equivalent and hence we can group them together to form a reduced graph. This equivalence will not, in general, preserve the lattice structure and hence we should ensure that our grouping equivalence relation on states is actually a congruence.

The interval clock approach is a specific example of a congruence on the distributive lattice of the computation which combines events on the same process. All consecutive events on a process that do not change the value of any of the variables that are relevant to the property being verified, are combined into one. This is clearly not the most general construction possible, though it is computationally very efficient. In general a process should report a new state to the monitor process only when it decides that it may possibly have changed the predicate being verified. We adapt Alagar and Venkatesan's interval clock approach accordingly. We then prove that the method of reducing the global state lattice using congruences works well with simple temporal formulae and then extend these results to the case of general nested temporal formulae. We show how this fits in with model checking and thus have a complete system for verifying arbitrary properties on distributed computations.

Our state space reduction approach differs from the *abstraction* based techniques used in model checking. Two common abstraction techniques are *cone of influence reduction* [3] and *data abstraction* [5]. Cone of influence reduction attempts to decrease the size of the graph, by focusing on the variables of the system that are referred to in the specification. Data abstraction, on the other hand, involves finding a mapping between the actual data values in the system and a small set of abstract data values. Both these techniques are different from the interval clock based state reduction and can indeed be used in conjugation with it. An important point of difference between the abstraction based techniques and the interval clock approach is that abstraction based techniques do not necessarily form *exact* approximations of the system. They can generate *false positives* during model checking which need to be tested out against the actual system. Our reduction is exact, in the sense that if a property holds in the original state lattice then it holds in the reduced state lattice and vice versa. If a property does not hold in the original state lattice then it also does not hold in the reduced state lattice and vice versa. The lattice property is indeed necessary for our state space reduction. We demonstrate this by a simple example where the global state graph is not a lattice in Figure 1. The black nodes represent states at which Φ holds. The property we are trying to verify is $definitely : \Phi$ (or, $AF(\Phi)$ in temporal logic CTL). It is clear from the figure that $definitely : \Phi$ holds in the original graph on the left but not on the reduced graph on the right.

System properties and specifications are assumed to be expressed in a restricted version of the *Computation Tree Logic (CTL)* [2] which does not have the *next-time* operator X. Next-time is not preserved by state reductions, and hence we focus on the remaining portion of the temporal logic,

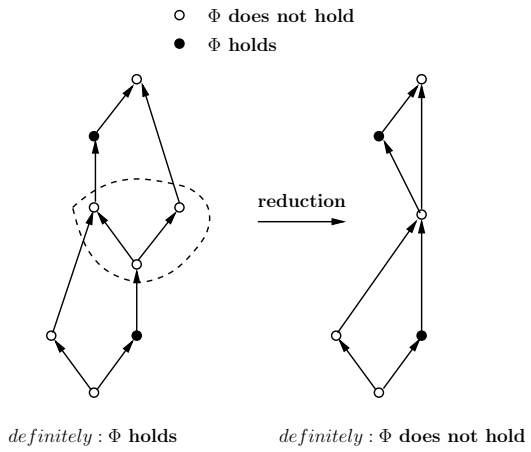


Figure 1: Lattice structure is necessary for our congruence-based state reduction. The figure demonstrates a case where the graph not being a lattice can lead to erroneous reductions.

denoted CTL_{-X}. This contains the operators EF, AF, EG, AG, EU and AU. $EF(\Phi)$ and $AF(\Phi)$ correspond to the *possibly* : Φ and $definitely : \Phi$ modalities respectively. *Linear Time Logics (LTL)* are also often used in model checking. Since the results we prove in the context of CTL_{-X} easily extend to LTL_{-X} (LTL without the X operator), hence we focus only on CTL_{-X} in this paper. Almost all the properties (like safety and liveness) that are of interest in distributed programs (esp. in the asynchronous environment) can be expressed in CTL_{-X}. For instance, in a mutual exclusion algorithm, the property "once a process requests a lock then it eventually gets the lock", can be expressed as $AG(request \Rightarrow AF(lock))$. As another example, "a ring of n processes always has exactly one token" can be expressed as $AG(token_1 + ... + token_n = 1)$.

The paper is organized as follows: We provide a brief background on our system model, lattice theoretic definitions and on the properties of congruences in Section 2. Our paper makes the following contributions:

- In Section 3 we show that reduction based on interval clocks is a special case of a congruence on the distributive lattice of the computation. Earlier [1] the interval clocks were updated whenever any variable related to the predicate Φ being detected was changed, even though the value of Φ itself may not have changed. We give methods to change the interval clock algorithm so that the interval clock gets updated only when an event on the process might change the value of Φ, thus resulting in greater reductions in the global state space.

- We show that simple (unnested) CTL_{-X} temporal formulae can be detected using the interval clock approach in Section 4.

- In Section 5 we extend these results so that we can handle nested temporal formulae and thus work with any arbitrary temporal formulae. We show how this fits in with model checking algorithms and present modifications required in model checking, in order to use our state reduction approach. The idea of interval clock based reduction integrates with any model checking algorithm since it does not assume anything about the internal workings of the model checking algorithm. We just change the global state graph that any model checking algorithm takes as input, to a much reduced global interval lattice.

- To the best of our knowledge, this is the first paper to express the problem of reducing the global state space of a distributed computation in terms of *congruences*. Consider the equivalence relation that groups together global states on the basis of the values of the properties that we are trying to detect in the computation. The problem of state reduction can then be viewed as the problem of coming up with a congruence that is contained in this equivalence relation. The

larger the congruence relation, the greater will be the state space reduction achieved. We utilize lattice theoretic properties of the distributive lattice of the computation and of congruences to derive our results. In Section 6 we present a centralized algorithm by which a monitor process can compute the optimal congruence for the most efficient global space reduction.

We conclude our work in Section 7. A brief background on CTL_{-X} is presented in the appendix.

2 Background

The execution of a single process in a computation results in a sequence of events totally ordered by the *occurred before* relation. We use $e <_p f$ to denote that e occurred before f on some process. To impose an order on events across processes, we use Lamport's happened-before relation $\rightarrow$ [12]. A distributed computation is defined as a partially ordered set (poset) consisting of the set of events P together with the happened before relation and denote it by $(P, \rightarrow)$. Two events e and f are concurrent in $(P, \rightarrow)$, (denoted $e \parallel f$), if $\neg(e \rightarrow f)$ and $\neg(f \rightarrow e)$. A global state (or a cut) is a subset $G \subseteq P$ such that $(f \in G) \wedge (e <_p f) \Rightarrow (e \in G)$. A consistent global state (CGS) of a computation $(P, \rightarrow)$ is a subset $G \subseteq P$ such that $(f \in G) \wedge (e \rightarrow f) \Rightarrow (e \in G)$.

A *global predicate* (or simply a *predicate* or *property*) is a boolean-valued function defined on the set of consistent global states. We say that $B(G)$ (B holds in CGS G) if the function evaluates to true in G. A *lattice* is a poset L such that for all $x, y \in L$, the least upper bound (*join*) of x and y (denoted $x \sqcup y$); and the greatest lower bound (*meet*) of x and y (denoted $x \sqcap y$) exists. A lattice L is *distributive* if for all $x, y, z \in L$: $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$. Given a computation P, we impose a partial order on the set of global states as follows: Given two consistent global states G and H, we say that $G \leq H$ iff $G \subseteq H$. Every lattice has a unique top element ($\top$) and a unique bottom element ($\bot$), such that if $x \in L$ then $\bot \leq x \leq \top$ (here $<$ and $\leq$ refer to the partial order relation on the lattice). We say that x *covers* y (denoted $x \prec y$), if $x < y$ and $\forall z \in L : (x \leq z < y) \Rightarrow (z = x)$. It is a well known result in lattice theory [8] that the set of all CGSs of a distributed computation forms a distributive lattice under the $\subseteq$ relation. In other words, any distributed computation corresponds to a distributive lattice of consistent global states of the computation.

A subset $S \subseteq L$, is a *sublattice* of L, iff S is non-empty and $\forall a, b \in S : (a \sqcup b) \in S$ and $(a \sqcap b) \in S$. Given two lattices L_1 and L_2, a function $f : L_1 \rightarrow L_2$ is said to be a *lattice homomorphism* if f is join and meet preserving, that is $\forall x, y \in L_1$, $f(x \sqcup y) = f(x) \sqcup f(y)$ and $f(x \sqcap y) = f(x) \sqcap f(y)$. An equivalence relation θ is called a *congruence* if it preserves the join and meet operations, that is

$$(x \equiv_\theta y) \Longleftrightarrow \forall z \in L : (z \sqcup x) \equiv_\theta (z \sqcup y)$$

$$and\ (z \sqcap x) \equiv_\theta (z \sqcap y)$$

A common notation for $(x \equiv_\theta y)$ is $x \equiv y(mod\theta)$. Lattice homomorphisms and congruences are closely related as stated by the following theorem:

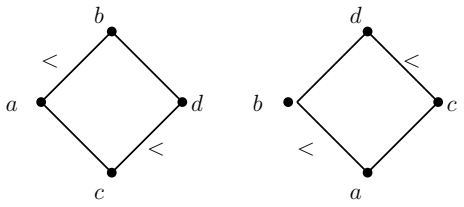
Theorem 1 *[8] Let L and K be lattices and let $f : L \rightarrow K$ be a lattice homomorphism. Then the equivalence relation θ defined on L by*

$$(\forall a, b \in L) :\ a \equiv b(mod\theta) \Longleftrightarrow f(a) = f(b)$$

is a congruence.

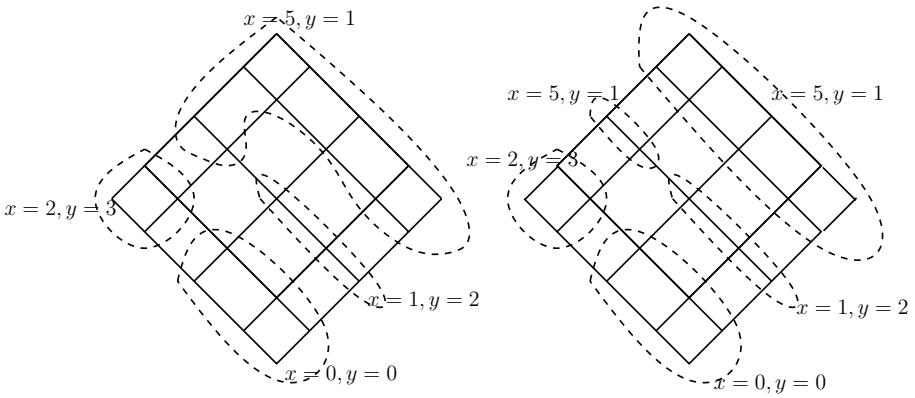
Congruences are often expressed by their corresponding lattice homomorphisms. In Section 3, we use this to show that the interval lattice construction is actually a congruence relation.

Given a congruence relation θ over a lattice L, the reduced lattice (also called the quotient lattice) is denoted by L/θ. Let $[a]_\theta$ be the equivalence class of elements equivalent to $a \in L$ under the congruence θ (also called *block* of θ). A block A of θ is said to be *convex* if $\forall x, y \in A$ and $\forall z \in L$, $(x \leq z \leq y) \Rightarrow (z \in A)$. Then $L/\theta = \{[a]_\theta | a \in L\}$.



quadilateral $< a, b; c, d >$

Figure 2: Two possible constructions of quadrilateral $< a, b; c, d >$ in lattice L.



Lattice partitioned into equivalence classes

Lattice partitioned into congruence classes

Figure 3: The problem of finding the largest congruence θ that is smaller than a given equivalence partition of the global state lattice L.

Suppose $a, b, c, d \in L$, then $< a, b; c, d >$ is said to be a *quadrilateral* if $a < b$, $c < d$ and either (1) $a \sqcup d = b$ and $a \sqcap d = c$, or (2) $b \sqcup c = d$ and $b \sqcap c = a$ (Figure 2). We say that blocks of congruence θ on L are *quadrilateral-closed* if whenever $a, b \in A$ and $A \in L/\theta$ then $\exists B \in L/\theta : c, d \in B$. We now state without proof a theorem on the structural characterization of congruences:

Theorem 2 *[8] Let L be a lattice and let θ be an equivalence relation on L. Then θ is a congruence if and only if (1) each block of θ is a sublattice of L, (2) each block of θ is convex, and (3) the blocks of θ are quadrilateral-closed.*

We are interested in grouping together global states which have the same state with respect to the property that we wish to verify. For example, if we are interested in detecting the property $(x^2+y > 10)$, then it would simplify the problem of detection if we can group together states that have the same x and y values. Doing this will induce an equivalence relation on the lattice and partition it into equivalence classes. However the structure formed by collapsing together the equivalence class elements does not in general form a lattice and thus does not represent a valid distributed computation. The reduced structure should also be a distributive lattice, in order for us to be able to apply other detection techniques on this reduced lattice. Congruences are equivalence relations that preserve distributivity in the reduced lattice [8]. The set of all congruences of a lattice also forms a lattice structure. Thus the *largest* congruence that is contained in a given equivalence relation is well defined. The problem of finding the greatest state space reduction possible is equivalent, to the problem of finding the largest congruence that is contained in the equivalence relation derived from the properties that we are trying to verify. Figure 3 illustrates this problem. The top most equivalence class in the first lattice is not a congruence class and hence it needs to be partitioned into two, to form a congruence class (second lattice). Note that the structure formed by collapsing the equivalence classes in the first lattice does not form a distributive lattice while it does so in the second lattice.

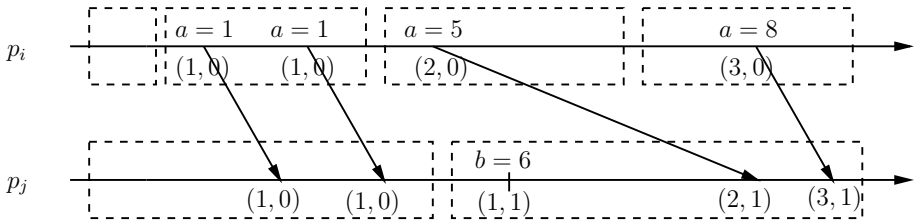


Figure 4: Intervals and interval clocks. Only variables a of p_i and b of p_j are relevant to the property Φ being detected.

3 Global Intervals and Interval Clocks

Alagar and Venkatesan [1] introduced the notion of Global intervals based on Interval clocks to reduce the global state lattice for detection of arbitrary properties under the *possibly* [7] modality. In their work, the monitor process is intimated of a new state whenever an event on a process changes the value of any variable related to the property being verified. However an event that changes the values of local variables related to a property Φ, does not necessarily change Φ. Hence these events can be merged together to form a single event on process p. For example, if we want to verify $\Phi \equiv (x \leq 7)$, then the value of x changing from 9 to 8 does not change Φ. We thus modify the Alagar-Venkatesan interval clock algorithm and later discuss how we can detect whether change of a local variable related to a property might actually change the property or not.

Definition 1 *An **interval** is a maximal sequence of consecutive events on a process such that property Φ being verified is the same after the occurrence of every event in the sequence.*

A process begins a new interval if an event can potentially change the value of Φ. Each process p_i maintains an *Interval Clock* V_i consisting of n components. Let the timestamp (ts) associated with a message be T. The interval clock has the following update rules:

1. **When p_i begins a new interval** (when Φ can potentially change):
2. $\quad V_i[i] = V_i[i] + 1$
3. **When p_i receives a message with ts T**:
4. $\quad \forall j : V_i[j] = max(V_i[j], T[j])$
5. **When p_i sends a message with ts T**:
6. $\quad T = V_i$

The timestamp of an interval I_i is denoted by $TS(I_i)$ and is the updated interval clock V_i when the first event of the interval I_i occurred. Figure 4 illustrates the concept of intervals and interval clocks. The interval clock differs from the traditional Fidge-Mattern vector clock [9, 13] because it does not increment the local component of the vector clock on every send and receive event. It does so only when an event on the process occurs which can potentially change Φ. It is also different from the *weak vector clock* of Marzullo and Neiger [14], in which a process increments its local component of the clock not only when an event on the process occurs which can potentially change Φ but also when a receive event occurs through which it perceives that another process has potentially changed Φ.

The key to changing intervals on a process is to decide whether the local event and the associated change of variables related to the property Φ, actually changed the value of Φ. Let the event be e and the property be $\Phi(x, y_1, ..., y_k)$ where $x, y_1, ..., y_k$ be the variables that Φ depends upon. Suppose that the event e changes the value of x to x'. The approach of Cooper and Marzullo using normal vector clocks would be to report a new global state on every event, irrespective of whether the property Φ or any of the variables $x, y_1, ..., y_k$ were changed or not. In Alagar and Venkatesan's approach, a new interval would be formed only when any of the variables $x, y_1, ..., y_k$ related to Φ were changed. However this does not necessarily mean that Φ changes. There are various examples where even though

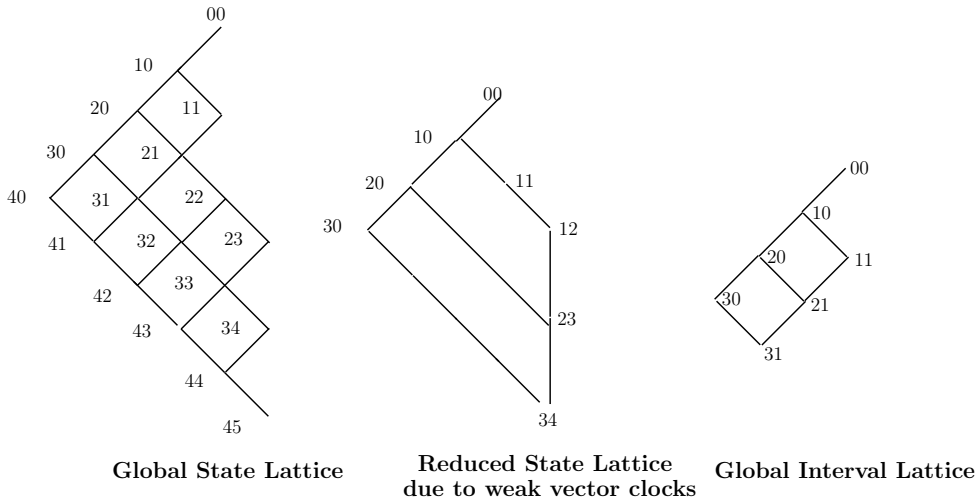


Figure 5: Global state lattices generated by various clock algorithms.

event e changed x to x', the value of Φ remains unchanged. For example, in the case of conjunctive predicates like, $\Phi \equiv (x = 5) \wedge (y_1 = 7) \wedge ... \wedge (y_k = 9)$, the value of x changing from 6 to 7 does not change Φ which remains $false$. As another example let x, y be variables on two different processes, and we are trying to detect the property that $\Phi \equiv (x + y \geq 75)$ always holds in the system. Then if Φ is already $true$ then increase in either variable will not change Φ. Similarly, if Φ is already $false$ then decrease in either variable will not change Φ. Thus there are various situations in which we can optimize further.

A general approach to do this optimization is to use *Binary Decision Diagrams (BDDs)* (or, *Multi-valued Decision Diagrams (MDDs)*) [6] to represent the property Φ being detected. Now, upon an event e, we want to check whether the values of $\Phi(x, y_1, ..., y_k)$ and $\Phi(x', y_1, ..., y_k)$ can be different. This can be verified by constructing the BDD for

$$\exists y_1 ... \exists y_k : (\Phi(x, y_1, ..., y_k) \wedge \neg\Phi(x', y_1, ..., y_k))$$

$$\vee(\neg\Phi(x, y_1, ..., y_k) \wedge \Phi(x', y_1, ..., y_k))$$

and checking to see whether this can evaluate to $true$. This can easily be extended to the case where the event e modifies more than one variable that occurs in the property. There are many algorithms that efficiently construct and evaluate these BDDs. This enables us to determine whether an event can potentially change the property Φ, and accordingly a new interval may be started.

We define two intervals to be consistent in the normal way: I_i and I_j are consistent iff the interval clock for each has a higher value for its respective component.

Definition 2 *[1] Two intervals I_i and I_j of processes p_i and p_j are said to be **consistent** if $TS(I_i)[j] \leq TS(I_j)[j]$ and $TS(I_j)[i] \leq TS(I_i)[i]$.*

Definition 3 *[1] A **global interval** is a collection of intervals with one interval from every process.*

Definition 4 *[1] A global interval $GI = (I_1, ..., I_n)$ is **consistent** if I_i and I_j are consistent for all i, j.*

Definition 5 *[1] A **global interval lattice** is the lattice formed by the set of all consistent global intervals with the $\leq$ order relation given by $I_i \leq I_j$ iff $\forall k \in [1, n] : I_i[k] \leq I_j[k]$.*

There is an edge from global interval I_i to I_j, if the computation can proceed from I_i to I_j by executing a sequence of events (interval) in a process. Figure 5 illustrates the global state lattice due to Fidge-Mattern vector clocks, reduced state lattice due to weak vector clocks and global interval

lattice corresponding to the computation shown in Figure 4. The states in the figure are labeled by the respective clock values. (Thus while 00 in each lattice corresponds to the initial state of the computation, in general the state labels do not correspond to the same state. For example, 11 refers to different states in the lattices). By comparing these lattices in the figure, it is clear that interval clocks generate a much reduced and compact representation of the lattice.

Alagar and Venkatesan proved that a predicate Φ is true at a global interval if Φ evaluates to true using the value of the variables related to Φ at the global interval. It is sufficient to test all the global intervals instead of all the global states to detect *possibly* : Φ for any arbitrary property Φ.

Theorem 3 *[1] There exists a global interval at which Φ is true if and only if there exists a consistent cut (global state) at which Φ is true.*

The global interval lattice can be looked upon as being generated from the global state lattice by a particular congruence θ given by the lattice homomorphism f (Theorem 1) characterized by:

1. $f(A \sqcup B) = f(A) \sqcup f(B)$
2. $f(A \sqcap B) = f(A) \sqcap f(B)$
3. $f(A) = f(B)$ iff $\forall i : TS(a_i)[i] = TS(b_i)[i]$

where $A = (a_1, ..., a_n)$ and $B = (b_1, ..., b_n)$ are global intervals of the computation. Thus if the original lattice is L and the reduced lattice L/θ, then $f : L \rightarrow L/\theta$. The first two statements are just the properties of a congruence (any congruence must preserve the join and meet operations). The third statement above represents the condition that two intervals A and B are mapped to the same congruence class if and only if the value of property Φ has not changed between the two global intervals.

Reducing a lattice by a congruence corresponds to grouping together elements of the original lattice. In terms of distributed computation, we can look upon this as combining events with the same value of Φ to form a smaller computation. The global interval technique is thus a special case of a congruence, where we combine appropriate consecutive events on each process in the system.

4 Detecting simple Temporal formulae using Interval Clocks

In this section we focus on the simple temporal logic properties $EF(\Phi)$, $AF(\Phi)$ and $E[\Phi U \Psi]$ where Φ, Ψ are non-temporal properties and prove that it is equivalent to detect these properties on either the reduced global interval lattice or the original global state lattice. (A brief background on CTL_{-X} is given in Appendix A). Later we will extend the results for nested temporal logic formulae and to the domain of model checking. The result in Theorem 3 [1] refers to detection of $EF(\Phi)$ on the global interval lattice. This is because *possibly* : Φ and $EF(\Phi)$ have the same semantics stating that there exists a state in the global state graph reachable from the initial state, at which Φ holds.

We now prove the same result for the other temporal logic properties in CTL_{-X}. The proofs for these are very different from the proof for EF, since EF refers only to the existence of a particular state in the global state graph which is easy to detect. AF and EU instead refer to paths in the global state graph and hence are harder to detect. We claim that it is sufficient to test all the states of a global interval lattice instead of all the states of a global state lattice to detect $AF(\Phi)$ and $E[\Phi U \Psi]$. We first note that the boolean value of a property remains the same within an interval. A consistent cut $C = (C_1, ..., C_n)$ is said to be contained within a consistent global interval $I = (I_1, ..., I_n)$ if

$$\forall i \in [1, n] : C_i \in I_i$$

Property 1 *Given a consistent global interval $I = (I_1, ..., I_n)$ and consistent cuts $C = (C_1, ..., C_n)$, $C' = (C'_1, ..., C'_n)$ of the global state lattice such that C, C' are contained in I, then it is not possible to have $\Phi(C)$ and $\neg\Phi(C')$.*

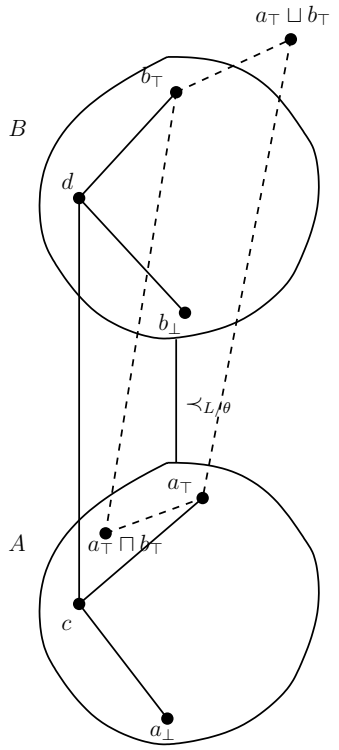


Figure 6: Proof of Lemma 1.

Consider a global state lattice L, the congruence θ and the reduced lattice L/θ. Then corresponding to every path P from bottom to top in L/θ, there is a corresponding path in the original lattice L from its bottom to top and vice versa. In the following we use $\prec_L$ to denote the covering relation in a lattice L. To prove the equivalence of paths between L and L/θ we need Lemma 2 which states that two states in the reduced graph have a covering relation between them if and only if they contain states which have a covering relation between them. Using this we prove the equivalence of paths in Lemma 3. We first need the following lemma:

Lemma 1 *Given two congruence classes A and B in L/θ, let $(a_\perp, a_\top)$ and $(b_\perp, b_\top)$ be the bottom and top element pairs of A and B respectively. If B covers A in L/θ, then there exists a path from $a_\top$ to $b_\top$ in L consisting only of nodes in A and B.*

Proof.

Since B covers A in L/θ, therefore there exist elements $c \in A$ and $d \in B$ such that $c \leq d$ in L (Figure 6). As $a_\perp \leq c$ and $d \leq b_\top$, we get

$$
\begin{aligned}
& a_\perp \leq b_\top \\
\Rightarrow\ & \{property\ of\ meet\} \\
& a_\perp \sqcap a_\top = a_\top \sqcap b_\top \\
\equiv\ & \{a_\perp \sqcap b_\top = a_\perp\} \\
& a_\perp \leq a_\top \sqcap b_\top \\
\equiv\ & \{property\ of\ meet\} \\
& a_\perp \leq a_\top \sqcap b_\top \leq a_\top \\
\Rightarrow\ & \{A\ is\ convex\} \\
& a_\top \sqcap b_\top \in A
\end{aligned}
$$

Therefore in Figure 6 $< b_\top, a_\top \sqcup b_\top; a_\top \sqcap b_\top, a_\top >$ forms a quadrilateral in L. Since $a_\top \sqcap b_\top \equiv a_\top (mod\ \theta)$ hence from the quadrilateral closed property of congruences we have, $b_\top \equiv a_\top \sqcup b_\top (mod\ \theta)$. Therefore, $a_\top \sqcup b_\top \in B$. By definition $b_\top$ is the top element of B and $b_\top \leq a_\top \sqcup b_\top$ implies that $b_\top = a_\top \sqcup b_\top$. Therefore $a_\top \leq b_\top$ (Connecting Lemma [8]). Hence there exists a path from $a_\top$ to $b_\top$ in L.

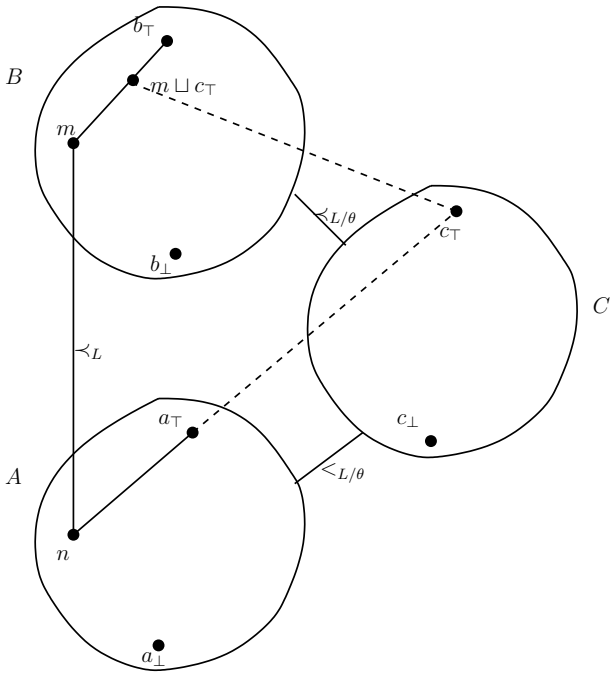


Figure 8(1)

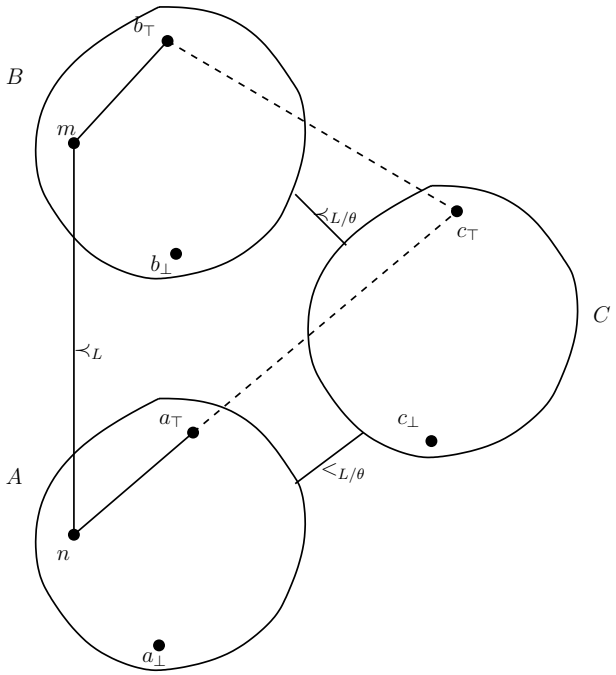


Figure 8(2)

Figure 7: Proof of Lemma 2.

It remains to be shown that the path consists only of nodes belonging to A or B. Pick any $e \in L$ such that it is on the path from $a_\top$ to $b_\top$. Thus $a_\top \leq e \leq b_\top$ and since $A \prec_{L/\theta} B$, hence by property of the covering relation either $e \in A$ or $e \in B$ which yields the desired result. □

Lemma 2 *Let A, $B \in L/\theta$, then*

$$A \prec_{L/\theta} B$$

if and only if there exists a, $b \in L$ such that

$$a \in A,\ b \in B \text{ and } a \prec_L b$$

Proof. The forward direction of the proof follows from Lemma 1 as follows: We assume that $A \prec_{L/\theta} B$. Thus by Lemma 1 there exists a path from $a_\top$ to $b_\top$ in L consisting only of nodes in A and B. The first element in the path is $a_\top$ and let the second element on the path be b. Clearly $b \in B$ and $a_\top \prec_L b$. Thus there exist m, $n \in L$ such that $n \in A,\ m \in B \text{ and } m \prec_L n$.

To prove the converse, we assume that $A,\ B \in L/\theta$ and there exists $m,\ n \in L$ such that $m \in A,\ n \in B \text{ and } m \prec_L n$. Let us assume that there exists $C \in L/\theta$ such that $A \prec_{L/\theta} C \prec_{L/\theta} B$ (Figure 7). We first note that as $m \prec_L n$, hence we cannot have $c_\top \leq m$ as there can only be one path from n to m in the covering graph (Figure 7(1)). Thus $\neg(c_\top \leq m)$ (Figure 7(2)). We first prove that $< m, m \sqcup c_\top; n, c_\top >$ forms a quadrilateral in L and then show that this contradicts our assumption about the existence of C. We first prove that $m \parallel c_\top$:

$C \prec_{L/\theta} B$
$\Rightarrow$ $\{Lemma\ 1\}$
$m \in B, b_\top \in B, c_\top < b_\top$
$\Rightarrow$ $\{Convexity\ of\ sublattice\ B, c_\top \notin B\}$
$\neg(m \leq c_\top)$
$\equiv$ $\{\neg(c_\top \leq m)\}$
$m \parallel c_\top$

Since $n \leq m$ and $m \leq b_\top$, by transitivity $n \leq b_\top$. Therefore, $< m, m \sqcup c_\top; n, c_\top >$ forms a quadrilateral

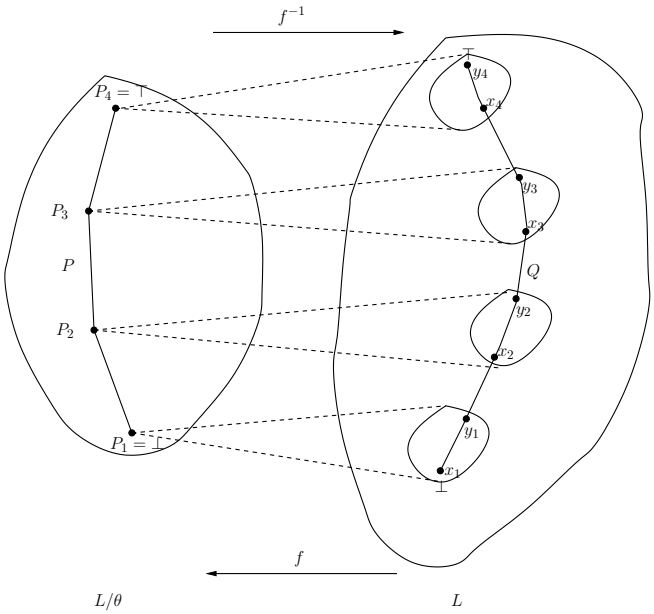


Figure 8: Proof of Lemma 3.

in L. Now we show that $m \sqcup c_\top \in B$. From the fact that B is a sublattice and Lemma 1, we get:

$m \leq b_\top$, $c_\top \leq b_\top$
$\equiv$ $\{property\ of\ join\}$
$m \sqcup c_\top \leq b_\top$
$\equiv$ $\{join\ property\}$
$m \leq m \sqcup c_\top \leq b_\top$
$\equiv$ $\{convexity\ of\ sublattice\ B\}$
$m \sqcup c_\top \in B$

Since $m \sqcup c_\top \equiv m(mod\ \theta)$ hence from the quadrilateral closed property of congruences we have, $n \equiv c_\top(mod\ \theta)$. Therefore, $c_\top \in A$. This contradicts our assumption that there exists $C \in L/\theta$ such that $A <_{L/\theta} C \prec_{L/\theta} B$. □

Now we can prove that there is a one-to-one correspondence between paths of L and L/θ with respect to the values of the variables that are relevant to the properties that we are trying to detect. The first part of the lemma says that for any path in L/θ, if we look at the pre-images of the nodes on the path (corresponding to inverse of the lattice homomorphism function f), then there exists a subset of these pre-image nodes which also forms a path in L. Since f is defined so as to preserve values of variables relevant to the property being detected, hence this enables us to prove the equivalence of detecting temporal formulae between L and L/θ. Similarly, the second part of the lemma proves that for every path in L, if we look at the image of the nodes on the path then they form a corresponding path in L/θ.

Lemma 3 *[Equivalence of Paths] Let f be the lattice homomorphism corresponding to θ.*

1. *$P = (P_1, ..., P_k)$ be a path from bottom to top in L/θ, then there exists a path Q in L such that $Q \subseteq \{q \in L : \exists i \in [1,k] : q \in f^{-1}(P_i)\}$.*

2. *$Q = (q_1, ..., q_k)$ be a path from bottom to top in L, then the set $P = \{f(q_1), ..., f(q_k)\}$ forms a path from bottom to top in L/θ (Figure 8 illustrates the lemma).*

Proof.

1. Consider the set $f^{-1}(P_i)$. For any $i \in [1,k]$, let x_i and y_i be the bottom and top elements of $f^{-1}(P_i)$ respectively ($f^{-1}(P_i)$ is a sublattice). Note that x_1 and y_k are the bottom and top

elements of L. From Lemma 1 we get that there is a path from $y_1 \rightarrow y_2 \rightarrow ... \rightarrow y_k$. Also since $f^{-1}(P_1)$ is a sublattice, hence there is a path $x_1 \rightarrow y_1$. Therefore there is a path from x_1 (bottom) to y_k (top) in L.

2. Let $f(q_{i,1}), ..., f(q_{i,l})$ be the sequence obtained from P after removing duplicate nodes. From the definition of $f(q_{i,k})$ and $f(q_{i,k+1})$, there exists m, $n \in L$ such that $m \in f(q_{i,k})$, $n \in f(q_{i,k+1})$ *and* $m \prec_L n$. Then Lemma 2, gives us that $f(q_{i,k}) \prec_{L/\theta} f(q_{i,k+1})$ and in general $f(q_{i,1}) \prec_{L/\theta} ... \prec_{L/\theta} f(q_{i,l})$ and thus this forms the desired path.

□

We can now show that it is sufficient to test all the states of a global interval lattice instead of all the states of a global state lattice to detect $AF(\Phi)$ and $E[\Phi U \Psi]$ for arbitrary non-temporal properties Φ, Ψ. $AF(\Phi)$ has the same semantics as $definitely : \Phi$ and is true if on all possible paths from $\perp$ to $\top$ in the computation, there is a state on which Φ is true. $E[\Phi U \Psi]$ means that there exists a path from $\perp$ to $\top$ such that Φ is true on all states on the path until a state is reached where Ψ is true. Let θ_I be the congruence on the global state lattice L induced by interval clocks and f be the corresponding lattice homomorphism. The reduced lattice is L/θ_I.

Theorem 4 *$AF(\Phi)$ holds in L iff $AF(\Phi)$ holds in L/θ_I.*

Proof.

1. $AF(\Phi)$ holds in $L \Rightarrow AF(\Phi)$ holds in L/θ_I

   Suppose that $AF(\Phi)$ does not hold in L/θ_I. Then there exists a path P in L/θ_I from its bottom to top element such that Φ is false on all interval states on that path. From Lemma 3-1, this corresponds to a path Q in the original lattice L. However since $AF(\Phi)$ holds in L, hence there exists a consistent cut/global state (say q) on this path such that $\Phi(q)$. Consider $f(q) \in L/\theta_I$. From our assumption, $\neg\Phi(f(q))$ in L/θ_I. This is a contradiction since the state of all the variables relevant to Φ are same at q and $f(q)$.

2. $AF(\Phi)$ holds in $L/\theta_I \Rightarrow AF(\Phi)$ holds in L

   Assume that $AF(\Phi)$ does not hold in L. Then there exists a path Q in L from its bottom to top element such that Φ is false on all global states on that path. From Lemma 3-2. there exists a path P from bottom to top in L/θ. Since $AF(\Phi)$ holds in L/θ_I, hence there exists an interval state (say I) on this path P such that $\Phi(I)$. Hence there exists a state $q \in I$ such that $\Phi(q)$. This is a contradiction since the state of all the variables relevant to Φ are same at q and I.

□

Theorem 5 *$E[\Phi U \Psi]$ holds in L iff $E[\Phi U \Psi]$ holds in L/θ_I.*

Proof.

1. $E[\Phi U \Psi]$ holds in $L \Rightarrow E[\Phi U \Psi]$ holds in L/θ_I

   Since $E[\Phi U \Psi]$ holds in L, there exists a path Q in L from its bottom to top element such that $(\Phi \wedge \neg\Psi)$ is true on all global states on that path until Ψ becomes true. Let the first state at which Ψ becomes true be q and the prior states on the path be $q_1, ..., q_k$. From Lemma 3-2. there exists a corresponding path P from bottom to top in L/θ. Hence on P, the set $\{f(q_1), ..., f(q_k)\}$ forms a sequence of nodes where $(\Phi \wedge \neg\Psi)$ holds and is followed by the node $f(q)$ where Ψ becomes true. Thus $E[\Phi U \Psi]$ holds in L/θ_I.

2. $E[\Phi U \Psi]$ holds in $L/\theta_I \Rightarrow E[\Phi U \Psi]$ holds in L

   Since $E[\Phi U \Psi]$ holds in L/θ_I, there exists a path P in L/θ_I from its bottom to top element such that $(\Phi \wedge \neg\Psi)$ is true on all global states on that path until Ψ becomes true. Let the first state at which Ψ becomes true be p and the prior states on the path be $p_1, ..., p_k$. From Lemma 3-1. there exists a corresponding path Q from bottom to top in L. Hence on Q, there exists a sequence of states $\{q_1, ..., q_k\}$ such that $\forall i \in [1, k] : q_i \in f^{-1}(p_i)$ and $(\Phi \wedge \neg\Psi)$ holds on each state in $\{q_1, ..., q_k\}$ and is followed by the node $q \in f^{-1}(p)$ where Ψ becomes true. Thus $E[\Phi U \Psi]$ holds in L.

□

5 Nested Temporal Predicates and Model Checking

We now extend our results to the other temporal properties in CTL_{-X} and also allow for arbitrary nesting of these properties. The only property of Φ (or Ψ) that we have actually used in our preceding arguments is:

Property 2 *The value of a property Φ at state s, $\Phi(s)$ can be evaluated using the values of variables at state s. In particular, we do not need to explore any other states in the graph.*

The temporal formulae for EG, AG and AU can be expressed in terms of EF, AF and EU. These equivalences are given in Appendix A. Using Property 2 and the temporal equivalences, the results for EF, AF and EU from the previous sections, readily generalize to EG, AG and AU.

We now look at handling nested temporal properties in CTL_{-X} in the model checking framework. We begin by constructing the reduced global interval lattice directly from the distributed computation. Instead of vector clocks, interval clocks are used so that new global states are added to the graph only when the properties we want to model check, can potentially change. The interval clocks are incremented with respect to the set of all non-temporal predicates embedded in the properties we need to verify. As an example, if we are trying to verify $AG(p \Rightarrow AF(r))$ and $AG(EF(p) \wedge q)$, then interval clocks will be based on the set $\{p, q, r\}$ of non-temporal predicates. A new interval is created when any of the predicates in this set can potentially change (Step 1 of the interval clock algorithm in Section 3).

Model checking algorithms [6] then evaluate nested temporal formulae on the global state graph by recursively evaluating all sub-formulae. Given the global interval graph G and the formula Φ, model checking algorithms will return the set of all states which satisfy Φ (say $[\Phi]$). Let *modelcheck*() be the function which does this. We modify this procedure so that along with returning $[\Phi]$, it also simultaneously labels each state s on the graph by whether Φ is true at s or not. (Note that this does not affect the time complexity of the model checking algorithm since in the worst-case, it has to visit all states in the state graph G. The actual running can be improved by using hash-tables instead of labeling.)

Using global interval lattices is now justified for nested temporal properties, since when we are trying to evaluate some nested formula Φ (eg. $\Phi = AG(EF(p) \wedge q)$), all its sub-formulae Ψ $(p, q, EF(p))$ satisfy Property 2. This is because when *modelcheck*() is called on Φ, it is first recursively evaluated at all its sub-formulae, during which each state in G gets labeled by the truth value of all these sub-formulae at that state. Thus in our example while model checking $\Phi = AG(EF(p) \wedge q)$, this acts as an unnested formula $AG(\Psi)$ since by looking at each state we already know whether or not $\Psi = EF(p) \wedge q$ holds at that state.

The overall approach can be summarized as:

1. *Find the set S of all sub-formulae without temporal operators, from the set of properties to be verified on the computation.*

2. *Create the global interval lattice L' from the computation by using interval clocks with respect to the set S.*

3. *Run model checking algorithm on L' with the modification that states are labeled in each step as described before. Nested temporal formulae, due to state labeling of sub-formulae, can be treated as simple unnested temporal formulae.*

This implies that it is sufficient to consider global intervals instead of global states to model check systems. In a process, the number of intervals can be considerably less than the total number of events, if every event does not change the properties of interest. Therefore, the total number of global intervals are likely to be substantially less than the total number of global states, thus improving the performance of algorithms to detect such properties. The idea of interval lattices also integrates with any model checking algorithm since it does not assume anything about how the model checking algorithm works. Instead, it just changes the global state graph to a global interval graph which the model checking algorithm takes as input.

6 Optimal Congruence

Using interval clocks, we were able to derive an online algorithm for state space reduction. The intervals are easy to compute by each process and hence each process was able to report only the relevant events to the monitor process. We also showed that this was not the optimal congruence that we could derive, since each process has to make a decision based on local information only. We now change the model to the following: each process now reports every event to the monitor process. The monitor process will have information from every process and will be able to compute exactly which global states need to be added to form the reduced global state lattice.

We borrow from the results of [10] (Chapter 2) to derive the optimal congruence algorithm. Given two elements $a, b \in L$, the smallest congruence that puts a and b in the same congruence class is called the principal congruence of a and b, denoted $Cg(a, b)$. We denote the set of join-irreducible elements of L by $J(L)$. If $x \in J(L)$, then we denote the unique lower cover of x by x_*. The optimal congruence is given by taking the join of relevant *principal* congruences.

Theorem 6 *[10] Given a lattice L and an equivalence relation E on L, the largest congruence that is contained in E is given by:*

$$\bigsqcup\{Cg(x, x_*) | x \in J(L), Cg(x, x_*) \subseteq E\}$$

We now convert this into an algorithm based on events reported to the monitor process. There is a one-to-one correspondence between events in the poset P of the distributed computation and the set of join-irreducible elements $J(L)$. If $e \in P$ corresponds to $x \in J(L)$, then x is the smallest global consistent cut that contains e and x_* is the same cut with the event e removed from it. The difference between x and x_* is the event e, hence $Cg(x, x_*) \subseteq E$ if and only if the event e on a global state x_* does not change the properties we wish to verify. The monitor process can easily verify this as it knows the global state x_* (which is obtained from the vector clock of e with the component corresponding to e reduced by 1). The monitor process, as it receives events from each process builds a *reduced* poset of the computation. For every event e that it receives, it first checks whether $Cg(x, x_*) \subseteq E$. If it does not hold, then e is simply added to the poset. Otherwise, the event e is omitted making sure that transitive relations through event e are maintained. Once the reduced poset is formed, the monitor process then constructs the reduced state lattice from it.

The algorithm for the monitor process is:

1. *Construct a poset P of the events reported by each process.*
2. *Construct a topological sort of the events reported by all processes.*
3. *For each event e in the topological sort:*
4. *Determine whether $Cg(x, x_*) \subseteq E$ where x is the least consistent cut containing e.*

5. *If $Cg(x, x_*) \subseteq E$ then remove e from P making sure that transitive relations are preserved.*

After generating the global state lattice from the reduced poset, it is ready to be used for model checking.

7 Conclusions

In this paper, we formulated the problem of reducing the global state lattice using congruences to deal with the state space explosion problem. The problem of state space reduction is equivalent to looking at the equivalence induced on the nodes of the global state graph by the value of the properties evaluated at each state, and then finding the largest congruence that is contained in this equivalence relation. We modified the concept of global intervals to yield smaller global state spaces. If every event in the system does not change the property being verified, then the global interval lattice will be much smaller than the total state space, leading to substantial improvements in the performance of verification algorithms.

We also presented an algorithm by which if all events are reported to the monitor process then it can compute the optimal congruence for our problem and achieve the greatest global state space reduction possible. Using lattice theoretic properties of distributed computations and congruences, we extended property verification using reduced lattices to the entire class of temporal logic formulae CTL_{-X} in the context of model checking. Even though we have dealt only with CTL_{-X}, our results readily extend to LTL_{-X} which is the other popular temporal logic used for model checking. Our state space reduction technique integrates seamlessly with any model checking algorithm.

References

[1] S. Alagar, S. Venkatesan. *Techniques to Tackle State Explosion in Global Predicate Detection.* In Proceedings of the IEEE Transactions on Software Engineering, Vol. 27, No. 8, Aug 2001, Pages 704 - 714.

[2] M. Ben-Ari, A. Pnueli, Z. Manna. *The Temporal Logic of Branching Time.* Acta Inf. 20, 1983, Pages 207 - 226.

[3] F. Balarin, A. Sangiovanni-Vincentelli. *An Iterative Approach to Language Containment.* In Proceedings of the 5th Workshop on Computer-Aided Verification, Jun/Jul 1993, Pages 29 - 40.

[4] E.M. Clarke, E.A. Emerson, A.P. Sistla. *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.* ACM Transactions on Programming Languages and Systems, Vol 8, No. 2, Apr 1986, Pages 244 - 263.

[5] E.M. Clarke, O. Grumberg, D.E. Long. *Model Checking and Abstraction.* ACM Transactions on Programming Languages and Systems, Vol. 16, No. 5, 1994, Pages 1512 - 1542.

[6] E.M. Clarke, O. Grumberg, D.A. Peled. *Model Checking.* The MIT Press, Cambridge, 1999.

[7] R. Cooper, K. Marzullo. *Consistent Detection of Global Predicates.* In Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, Santa Cruz, California, 1991, Pages 163 - 173.

[8] B.A. Davey, H.A. Priestley. *Introductions to Lattices and Order.* Cambridge Mathematical Textbooks, Cambridge University Press, 1990.

[9] C. Fidge. *Timestamps in message-passing systems that preserve the partial ordering.* In Proceedings of the 11th Australian Computer Science Conference, 1988, Pages 56 - 66.

[10] R. Freese, J. Jezek, J.B. Nation. *Free Lattices.* Mathematical Surveys and Monographs, Vol. 42, American Mathematical Society, Providence, RI, 1995.

[11] V.K. Garg, B. Waldecker. *Detection of Unstable Predicates in Distributed Programs.* In Proceedings of the 12th Conference on the Foundations of Software Technology and Theoretical Computer Science, New Delhi, India, Lecture Notes in Computer Science, 652, Springer Verlag, Dec 1992, Pages 253 - 264.

[12] L. Lamport. *Time, Clocks and the Ordering of Events in a Distributed System.* Communications of the ACM, Vol. 21, No. 7, 1978, Pages 558 - 564.

[13] F. Mattern. *Virtual Time and Global States of Distributed Systems.* In Proceedings of the International Workshop on Parallel and Distributed Algorithms, North-Holland, 1989, Pages 120 - 131.

[14] K. Marzullo, G. Neiger. *Detection of Global Stable Predicates.* In Proceedings of the Fifth Workshop on Distributed Algorithms and Graphs, Oct 1991, Pages 254 - 272.

A Temporal Logic CTL_{-X}

The specification language that we use is a subset of the propositional, branching-time temporal logic called Computation tree logic (CTL) [2]. We focus on the subset of CTL that does not include the next-time operator (X), denoted by CTL_{-X}. The formal syntax of CTL_{-X} is given below. AP is the set of atomic propositions.

1. Every atomic proposition $p \in AP$ is a CTL_{-X} formula.
2. If Φ and Ψ are CTL_{-X} formulae, then so are $\neg\Phi$, $\Phi \wedge \Psi$, $EF(\Phi)$, $EG(\Phi)$, $AF(\Phi)$, $AG(\Phi)$, $E[\Phi U \Psi]$, $A[\Phi U \Psi]$.

The semantics of CTL_{-X} is defined with respect to a labeled state transition graph. A CTL_{-X} structure is a triple $M = (S, R, P)$ where S is the finite set of states, R is the total binary relation on S which gives all the possible transitions between states and $P : S \rightarrow 2^{AP}$ assigns to each state the set of atomic propositions true in that state. A *path* is an infinite sequence of states $(s_0, s_1, ...)$ such that $\forall i : (s_i, s_{i+1}) \in R$. For any structure $M = (S, R, P)$ and a state $s_0 \in S$, there is an infinite computation tree with root labeled s_0 such that $s \rightarrow t$ is an arc in the tree iff $(s, t) \in R$. The notation $M, s_0 \models f$ means that formula f is true at state s_0 in the structure M. Thus we have:

1. $M, s_0 \models p$ iff $p \in P(s_0)$
2. $M, s_0 \models \neg\Phi$ iff $\neg(M, s_0 \models \Phi)$
3. $M, s_0 \models \Phi \wedge \Psi$ iff $M, s_0 \models \Phi$ and $M, s_0 \models \Psi$
4. $M, s_0 \models A[\Phi U \Psi]$ iff for all paths $(s_0, s_1, ...)$, $\exists i : i \geq 0$ and $M, s_i \models \Psi$ and $\forall j : j \in [0, i) : M, s_j \models \Phi$
5. $M, s_0 \models E[\Phi U \Psi]$ iff for some path $(s_0, s_1, ...)$, $\exists i : i \geq 0$ and $M, s_i \models \Psi$ and $\forall j : j \in [0, i) : M, s_j \models \Phi$

The other CTL_{-X} formulae can be written as:

1. $AF(\Phi) \equiv A[trueU\Phi]$ which means that Φ holds sometime in the future along every path from s_0.
2. $EF(\Phi) \equiv E[trueU\Phi]$ which means that there is some path from s_0 that leads to a state at which Φ holds.
3. $EG(\Phi) \equiv \neg AF(\neg\Phi)$ which means that there is some path from s_0 on which Φ holds at every state.
4. $AG(\Phi) \equiv \neg EF(\neg\Phi)$ which means that Φ holds at every state on every path from s_0.

In Section 5, we use the following equivalences to generalize our results from EF, AF and EU to EG, AG and AU:

1. $EG(\Phi) \equiv \neg AF(\neg\Phi)$
2. $AG(\Phi) \equiv \neg EF(\neg\Phi)$
3. $A[\Phi U \Psi] \equiv \neg EG(\neg\Psi) \wedge \neg E[\neg\Psi U(\neg\Phi \wedge \neg\Psi)]$