# Checking properties of distributed computations

Jyotirmoy V. Deshmukh

Department of Computer Sciences and Computer Engineering Research Center,

The University of Texas at Austin, Austin TX 78712, USA

deshmukh@cs.utexas.edu

**Abstract.** Predicate detection is an important problem in distributed systems. Predicate detection suffers from state explosion since the number of possible global states is exponential in the number of processes. Computation slicing is an important abstraction technique used to solve the predicate detection problem. The key observation in this paper is that an equivalence relation on the global state lattices for the distributed program can be translated to a corresponding equivalence relation on the respective posets. Thus detecting a predicate on a smaller poset is enough to make sure that it is satisfied by a larger equivalent poset. This reduces the complexity of using the slicing based approach as the size of the poset is smaller. We show the use of stuttering bisimulation as an effective abstraction technique for reasoning about conjunctive predicates.

## 1 Introduction

Given a distributed system, we are often interested in checking whether the system satisfies certain properties. A distributed computation represents one possible *run* of the distributed system. Given a distributed computation, detecting whether a predicate holds true for the computation can be achieved by model checking the global state space of the computation. However, this approach faces combinatorial explosion in the number of states, since the number of global states is exponential in the number of processes.

There are various techniques prevalent to reduce the size of the global state space. Symbolic model checking using BDDs, [McMill '92], predicate abstraction, [GrSa '97] symmetry reductions, [ES '96], and counter example guided abstraction refinement, [Clarke et al. '00], are common techniques that have been used to reduce the size of the state space in the traditional model checking domain.

Loosely coupled distributed systems with message passing have the nice property that the set of global states forms a distributive lattice with the happened before relation. Abstractions on the lattice of global states have been effectively used to reduce the size of the state space, [ChGa '04]. The biggest disadvantage

of such abstraction techniques lies in the construction of the global state graph, which can be exponentially large.

Computation slicing is an important abstraction technique used to solve the predicate detection problem for distributed systems, [MG '01], [SeGa '03]. A computation *slice* defined with respect to a global predicate is the computation with the least number of global states that contains all the global states of the original computation for which the predicate evaluates to true. Slicing throws away global states which are not relevant to the predicate being detected. Thus a much more efficient search of the state space can be achieved by slicing the computation. The biggest advantage of the slicing based approach is that it directly works off the partially ordered set of events, and does not require the construction of a global state graph.

In this paper we show how we can use equivalence relations on the lattice of global states to perform abstraction. Moreover we show that for certain classes of predicates, a grouping of global states into equivalence classes results in the grouping of corresponding local states into equivalence classes under some relation. Since the predicates we consider cannot be distinguished by equivalent lattices, they also cannot be distinguished by corresponding equivalent posets, by our previous claim. Thus verification of a predicate on a larger poset can be reduced to checking the predicate over a smaller reduced poset. We can then apply the slicing technique to this smaller reduced poset, and thus perform the verification without needing to build the global state graph.

We assume that the system properties (predicates) are expressed in a restricted version of the branching time logic, $CTL^*$ without the next time operator $X$, denoted as $CTL^*_{-X}$. Next-time operator is not preserved by grouping consecutive sets of states, and hence we focus on the remaining portion of the temporal logic. Many interesting properties such as *safety*, *liveness* and *fairness* can be expressed in this logic.

The remainder of the paper is organized as follows. In Section 2 we briefly present the computation model to represent distributed programs. We present the idea for obtaining equivalence relations on posets in Section 3. We show how we can use abstractions on lattices to perform a corresponding abstraction on the respective poset in Section 4. We have a look at some of the related work in Section 5. Finally, some concluding remarks and future directions of work are provided in Section 6.

## 2 Preliminaries

In this section we introduce the computation model to be used. We also provide an overview of the class of predicates we consider for verification.

We use a computation model similar to the one presented in [SeGa '03]. We assume a loosely-coupled message-passing asynchronous system without any shared memory or a global clock. We assume that the distributed system is comprised of $n$ processes $P_1, \ldots P_n$ which can each either execute an *internal* event, or *send* a message or *receive* a message. We do not make any assumptions about FIFO ordering of messages in the channel.

The distributed system is modeled as a partially ordered set of events, $G = \langle E, \rightarrow \rangle$, where $\rightarrow$ denotes the *happened before* relation. $G$ is a directed graph with vertices as events $e_i \in E$ and edges as $\rightarrow$. Two events $e$ and $f$ are termed concurrent in $G$, denoted ($e \parallel f$) if $\neg(e \rightarrow f) \wedge \neg(f \rightarrow e)$.

We define a *consistent cut* (global state) on directed graphs as a subset $J \subseteq E$ such that $e_i \in J \wedge e_i \rightarrow e_j \Rightarrow e_i \in J$. A consistent cut captures the notion of a reachable global state. The set of all consistent cuts for a distributed computation, $C(E)$, forms a lattice $\mathcal{L} = (C(E), \subseteq)$. The *join* operator, $\sqcup$, is the set union and the *meet* operator, $\sqcap$, corresponds to set intersection. The set of maximal events with respect to $\rightarrow$ of a consistent cut $J$ are denoted by $frontier(J)$. We denote the $initial$ cut of the computation by $\phi$ and the $final$ cut by $\varepsilon$.

A *global* predicate $\varphi$ is a boolean-valued function defined on the set of consistent global states. We say that $s \vDash \varphi$ or $\varphi(s)$ if the function evaluates to true for the global state $s$. A local predicate is a boolean-valued function defined on the set of all states local to a process. A conjunctive predicate has the form $\varphi = l_1 \wedge l_2 \wedge \ldots l_n$, where each of the $l_i$ is a local predicate. A conjunctive predicate is the special case of a regular predicate, which has the property that if $\varphi(s) \wedge \varphi(t)$ is $true$ then, $\varphi(s \sqcap t)$ and $\varphi(s \sqcup t)$ also hold $true$. Here $\sqcap$ and $\sqcup$ denote the *meet* and *join* operators respectively. A regular predicate has the special property that the set of all global states satisfying the predicate form a *sublattice* of the global state lattice.

In this paper we focus chiefly on conjunctive predicates. We can also reason about certain regular predicates, however no obvious solution seems possible for the application of our technique to relational pred-

icates. A relational predicate is of the form $a_1 * x_1 + a_2 * x_2 + \ldots a_n * x_n > c$, where each $x_i$ is a local variable of each process and $a_1, \ldots a_n$ and $c$ are constants. [1]

## 3 Abstraction techniques

In this section we present the abstraction technique to reduce the size of the global state graph. Firstly, we introduce some terminology.

### 3.1 Equivalence relations on graphs

We model the partially ordered set of events as a directed acyclic graph $G(V, E)$. We define a labeling function $L_{poset} : V \rightarrow 2_{AP}$ where $AP$ is the set of atomic propositions. Thus each event (state) in the poset is associated with a set of *local* predicates that are true for that event (state).

As seen in Section 2, a predicate is a function $P : AP \rightarrow 0, 1$. Let $Pred$ denote the set of all such predicates. We treat the lattice $\mathcal{L}$ of global states as a Kripke structure $(S, R, L_{lat}, s_0)$, where:

- $S$ is the set of global states,
- $L_{lat}$ is a labeling function $L_{lat} : S \rightarrow 2^{Pred}$,
- $s_0$ is the initial state, corresponding to the minimum cut ($inf$ of the lattice).

.

Given two Kripke structures, we can define various homomorphism relations between them. We apply the definitions of bisimulation and stuttering bisimulation in [EJP '97] to directed acyclic graphs representing partially ordered set of events. We can similarly define simulation and stuttering simulation relations on such graphs.

A path $\pi$ in a graph is a sequence of vertices of the graph of the form $\langle v_0, \ldots, v_{n-1} \rangle$. Since the graphs we consider are finite graphs, we consider only finite paths. We also assume that a graph $G$ has a unique starting node called $root_G$.

Consider two graphs $G(V, E)$ and $G'(V', E')$. A relation $\mathcal{B} : V \rightarrow V'$ is called a bisimulation relation iff the following conditions hold:

[1] Relational predicates are regular only if certain monotonicity condition is imposed on the variables $x_i$.

– $(root_G, root_{G'}) \in \mathcal{B}$
– Assume that $(v, v') \in \mathcal{B}$. Then the following conditions hold:
  - $L(v) = L(v')$
  - If $(v, w) \in E$, $L(v) = L(v')$, there exists $w' \in V'$ such that $(w, w') \in E'$.
  - A symmetric condition holds true with the roles of $v$ and $v'$ reversed.

If the last condition is relaxed, then the relation is called a *simulation* relation. In this case, we say that the graph $G'$ simulates the graph $G$. A bisimulation imposes an exact path correspondence and hence is an exact abstraction technique. An abstraction obtained by a simulation relation is a conservative abstraction. This means that if there is no path satisfying a particular predicate in the abstract structure, then there exists no such path in the concrete structure. However existence of such a path in the abstract structure, does not imply its existence in the concrete structure.

Another exact abstraction function can be obtained by considering paths which are equivalent up to the stuttering states. We define a stuttering path correspondence as follows:

Consider a relation $\mathcal{B}_{st} \subseteq V \times V'$. Two paths $\pi = \langle v_0, \ldots v_m \rangle$ and $\pi' = \langle w_0, \ldots w_n$ are stuttering $\mathcal{B}_{st}$-equivalent, iff there exists finite sequences of natural numbers $q = i_1 < \ldots < i_n$ and $q' = k_1 < \ldots < k_m$ such that for all $j \geq 0$ the following condition holds: $i_j < r < i_{j+1}$ and $k_j < p < k_{j+1}$, $(s_r, t_p) \in \mathcal{B}_{st}$
Consider two graphs $G(V, E)$ and $G(V', E')$. We say that a relation $\mathcal{B}_{st} \in V \times V'$ is a *stuttering bisimulation* iff:

– $(root_G, root_{G'}) \in \mathcal{B}_{st}$
– If $(v, v') \in \mathcal{B}_{st}$, $L(v) = L(v')$ and for every path $\pi$ starting from $v$ in $G$, there exists a stuttering $\mathcal{B}_{st}$-equivalent path starting from $v'$ in $G'$.
– $\mathcal{B}_{st}$ is symmetric.

If the last condition is relaxed the relation is termed as a *stuttering simulation.*

### 3.2 Abstraction technique

A key technique used in model checking to ameliorate state explosion is to abstract the state space by imposing an equivalence relation on the states. An equivalence relation groups certain states into an equivalence class. Under such an equivalence relation (or a homomorphism) $\rho$, we compute a quotient structure

or an abstract structure $H\backslash\rho$ by mapping all equivalent states to a single abstract state. We employ a similar technique by introducing equivalences on lattices.

Consider the lattice, $\mathcal{L}$ of consistent cuts of a distributed computation, $P$. We can obtain an infinite family of lattices $\mathcal{F}$ which has the property that for each lattice $\mathcal{K} \in \mathcal{F}, (\mathcal{K}, \mathcal{L}) \in \mathcal{B}_{st}$. Here $\mathcal{B}_{st}$ is a stuttering bisimulation relation. Given such a family of lattices there exists a smallest lattice stuttering bisimilar to all lattices in the family. Such a lattice can be obtained by grouping stuttering bisimilar states. This is similar to quotient construction technique in model checking. We term the smallest such lattice as the *cut-off* lattice $\mathcal{L}_c$. This is similar to the coarsest bisimulation construction technique.

*Remark*: The smallest structure obtained by abstracting a lattice based on the labels of the states in the lattice may not be a lattice. Hence we consider only the smallest *lattice* equivalent to the (concrete) global state lattice.

It can be shown that $CTL^*_{-X}$ properties are preserved by a stuttering bisimulation, [BCG '88]. Thus the predicate detection problem can be reduced to checking the predicate over the smaller global state space as represented by the cut-off lattice.

## 4 Reduction of Posets

In this section we show how we can use reductions on the global state space graph to perform a corresponding reduction on the poset. The reduction technique is illustrated with an example.

**Theorem 1.** *Given two stuttering bisimilar partially ordered sets representing distributed computations, the global state lattices, (with the set of predicates limited to conjunctive predicates), corresponding to the posets are stuttering bisimilar and vice versa.*

*Proof.* The proof follows an inductive argument on the definition of a stuttering bisimulation. A conjunctive predicate is of the form $\varphi = l_1 \wedge l_2 \ldots \wedge l_n$. Two global states are stuttering equivalent if they are labeled with the same predicate. For a conjunctive predicate, we know that,

$$\varphi \Rightarrow l_1, \ldots, \varphi \Rightarrow l_n \tag{1}$$

We consider two global state lattices $\mathcal{L}$ and $\mathcal{L}'$. Let the respective posets be $P$ and $P'$. We assume that each state of the lattices is labelled with a conjunctive predicate. If local state $s$ on process $i$ is part of the global state $K$, and $s'$ on process $i$ is part of the global state $K'$, and $K, K'$ are stuttering bisimilar, then:

- From (1), we know that $\varphi(K) \Rightarrow l_i(s)$ and $\varphi(K') \Rightarrow l_i(s')$. Thus if $L_{lat}(K) = L_{lat}(K')$, then $L_{poset}(s) = L_{poset}(s')$.
- Suppose global states $K$ and $K'$ are stuttering bisimilar. Thus for every path from $K$ in $\mathcal{L}$, there exists a stuttering equivalent path from $K'$ in $\mathcal{L}'$. Consider the local state on process $i$ for global state $K$ to be $s$. When the global state $K$ is advanced to the next global state, then it gets advanced either along process $i$ or the next global state also contains $s$.

  In the first case, if the next global state is stuttering equivalent to $K$, then the value of the local predicate should remain unchanged at the next local state along process $i$ (since it is a conjunctive predicate). If the next global state does not have the same label as $K$ (not stutt. eq.), then it can be either due to the local predicate at process $i$ (in which case the local state will get a different label) or due to local predicate at some other process. In the latter case the local predicate on process $i$ can remain the same.

  In the second case, the local state is trivially equivalent in both the posets as it is the same state. Thus, if there exists a stuttering path correspondence in the global state lattices, there exists a stuttering path correspondence in the posets.

Thus, by the definition of stuttering bisimulation, since every state in $P$ is stuttering bisimilar to some state in $P'$, $P$ and $P'$ are stuttering bisimilar.

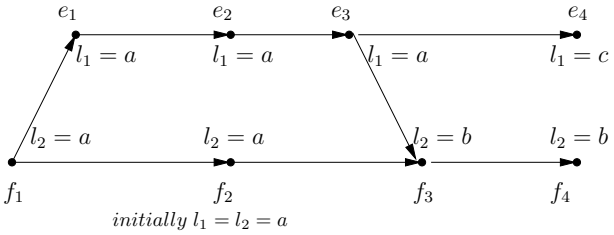
□

Thus, given a poset, we can reduce it to a smaller poset which is stuttering bisimilar to it. It can be shown that a stuttering bisimulation preserves $CTL^*_{-X}$ properties. Thus in order to detect such a property, it is sufficient to check it on such a reduced poset of a smaller size. This considerably reduces the complexity of the slicing algorithm, as the number of events in the poset is considerably reduced.

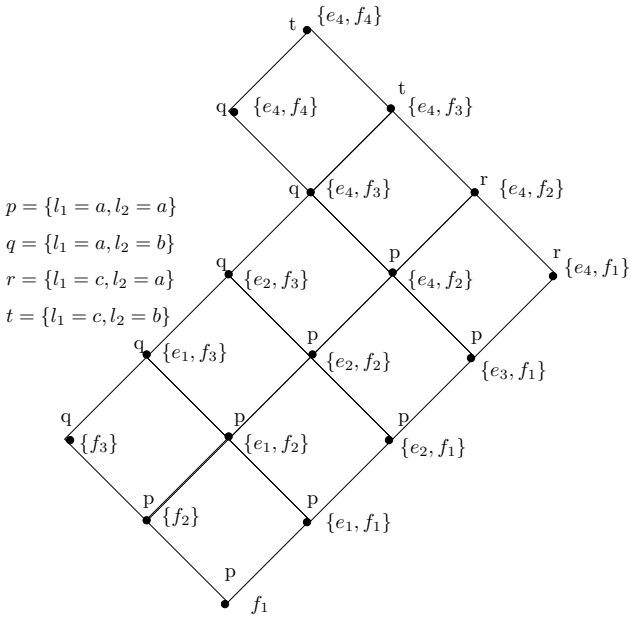
We illustrate our technique with the help of an example:

**Example**

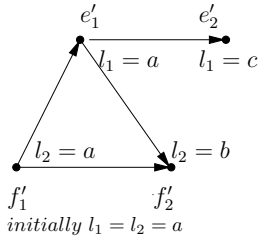
The abstraction technique is shown in Figure 1. We are given a poset $P$ with each event labeled with a



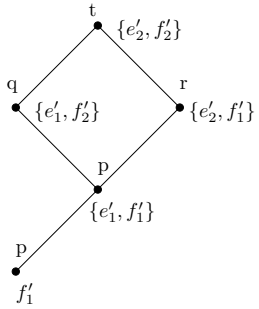

(a) Poset ($P$) representing a distributed computation




(b) Lattice corresponding to global states of Poset in (a)




(c) Reduced Poset Q




(d) Lattice corresponding to global states of Poset in (c)

**Fig. 1.** Abstraction using stuttering bisimulation

set of atomic propositions, as shown in Figure 1 (a). We compute the global state lattice. In this example, we consider four conjunctive predicates $p, q, r, t$. A predicate we might be interested in checking over the global state space could be of the form $\mathbf{EF}t$ or $\mathbf{EF}\ \mathbf{EG}q$. It can be clearly seen that checking these predicates over the lattice shown in Fig.reduction(b) is equivalent to checking the predicates over lattice shown in Fig.reduction(d). Also, the lattice in Fig.reduction(d) corresponds to the reduced poset as shown in Fig.reduction(c).

### 4.1 Other regular predicates

Some channel predicates or a predicate of the type ,"*There is no token message in transit*" can be checked using our technique. This is possible as there is a stuttering of the global states that can also be observed locally in the distributed computation. For example for the *no token in transit* predicate, we can group all consecutive states (possibly on different processes) which have the *token*. Relational predicates are tougher to handle, since a relational predicate $\psi$ on a set of local variables $x_1 \ldots x_n$, cannot be broken down into predicates that are locally true. For reasoning about relational predicates, we can use a conservative abstraction. A relational predicate can be regular only if there is a certain monotonicity constraint on the local variables. We can group local variables together if their values do not differ by some heuristic number. For instance if the predicate we are interested in detecting is $x_1 - x_2 > 0$, then we can decompose this predicate into local predicates $x_1 > \lambda$, $\lambda > \delta$, $\delta > x_2$, where $\lambda$ and $\delta$ are heuristically chosen for every group of states. However this is a conservative abstraction.

## 5 Related work

There are several techniques that can be used to reason about sets of partial order traces. The theory of Petri Nets offers a rich formalism to reason about the correctness of a distributed system. However many problems in Petri Net theory are undecidable in general. Using automata to check properties of partial order traces has also been studied. Mazurkiewicz trace theory is another rich formalism that is used to model concurrent systems. By using a notion of independence of actions, equivalence relations on traces can be expressed in this theory [Leu '02]. An automata theoretic approach can be then used to check whether a particular trace is

a member of a trace language. Checking non emptiness of Linear Büchi Alternating automata can however face an exponential blow-up in the size of the automaton.

In [ChGa '04] global state space reduction is performed using lattice homomorphisms. A lattice homomorphism is a join and meet preserving function between two lattices. A congruence on a set of states is a join and meet preserving equivalence relation. A lattice homomorphism $f$, induces a congruence relation on the set of states, in which congruent states are grouped together and all such states $s$ get mapped to the state $f(s)$. The authors explain how events on a single process can be combined together to obtain to obtain a reduced state graph. The function used for combining events locally on a process is a lattice homomorphism, and efficient techniques for computing optimal congruence are provided. Our method differs from the work in this paper in two aspects. Firstly, our abstraction may not be in general a lattice homomorphism. However, in order to obtain a reduction of the corresponding poset, we consider only abstract structures which are lattices. Furthermore, with our technique we can combine events on arbitrary processes as long as there is a stuttering path correspondence. Finally, our method can work directly off the poset, and does not require the construction of the global state graph.

## 6 Conclusions and future work

We present an abstraction technique for predicate detection in a distributed system. Predicate detection in a distributed computation, modeled as a poset, can be performed using slicing techniques. However in order to reduce the complexity of the slicing technique, we can reduce the size of the poset by a suitable abstraction. If the global state lattices for two distributed computations are stuttering bisimilar, we can show that the two corresponding partially ordered sets a also exhibit a stuttering bisimulation equivalence. We consider the set of conjunctive predicates, though our technique can be applied to few other regular predicates.

One main assumption for a slicing based approach is that we are given a single poset representing a single *run* of the distributed system. Thus slicing works well for run-time checking of predicates for a distributed system. However in order to prove a certain property of the distributed system, we should be able to prove the property for *all runs* of the system. Since each run corresponds to a distributed computation represented as a partially ordered set (poset), we need to check whether a property is satisfied by all posets that can be generated by the distributed system. This problem is much harder as inherent non-determinism

in message passing between processes can give rise to an exponential (or possibly unbounded) number of posets corresponding to a distributed system (If there are no restrictions on the transit time for messages, etc.).

In order to tackle this problem, we can extend the slicing based approach to handle a set of posets instead of a single poset. The main idea would be to define an equivalence relation on the set of global state lattices. The predicates we consider will have the property that they cannot distinguish between equivalent structures. Furthermore such an equivalence relation would correspond to an equivalence relation on the set of the respective posets. Given such an equivalence relation, the predicate detection problem for all posets would reduce to checking the predicate only over all the equivalence classes.

## References


[BCG '88] Michael C. Browne, Edmund M. Clarke, Orna Grumberg, *Characterizing finite Kripke structures in propositional temporal logic*, Theoretical Computer Science, vol. 59, pp. 115-131, 1988.

[McMill '92] K. L. McMillan. *Symbolic model checking - an approach to the state explosion problem*, PhD thesis, Carnegie Mellon University, 1992.

[Em '90] E. A. Emerson, *Temporal and modal logic*, In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, Elsevier, (1990)

[SeGa '03] Alper Sen and Vijay K. Garg, *Detecting Temporal Logic Predicates in Distributed Programs Using Computation Slicing*, 7th International Conference on Principles of Distributed Systems, La Martinique, France, December 10-13 2003.

[MG '01] Neeraj Mittal and Vijay K. Garg, *Computation Slicing: Techniques and Theory*, 5th International Symposium on DIStributed Computing (DISC'01), pp. 78-92.

[SeGa] Alper Sen and Vijay K. Garg, *Automatic Generation of Slices for Temporal Logic Predicate Detection*, Technical Report. Available at: http://maple.ece.utexas.edu/TechReports/2003/TR-PDS-2003-001.ps

[EJP '97] E. Allen Emerson, Somesh Jha and Doron Peled. *Combining Partial Order and Symmetry Reductions*, TACAS 1997: Pages 19-34.

[MT '96] . Mukund and P. S. Thiagarajan, *Linear Time Temporal Logics over Mazurkiewicz Traces*, Proc. MFCS '96, Springer LNCS 1113, (1996) 32-62.

[Leu '02] M. Leucker, *Logics for mazurkiewicz traces*, Technical Report AIB-2002-10, RWTH, Aachen, Germany, April 2002.

[FS '01] ernd Finkbeiner, Henny Sipma, *Checking Finite Traces using Alternating Automata*, Electronic Notes in Theoretical Computer Science 55(2), 2001.

[ES '96] E. A. Emerson and A.P. Sistla, *Symmetry and Model Checking*, Formal Methods in System Design: An International Journal, Vol 9., 1996, Kluwer Academic Publishers, pp.105-131.

[GrSa ’97] S. Graf and H. Saidi, *Construction of abstract state graphs with PVS*, In CAV. Springer-Verlag, June 1997.

[ChGa ’04] Arindam Chakraborty and Vijay K. Garg, *On Reducing the Global State Graph for Verification of Distributed Computations*, To be published, Available as Technical Report no. TR-PDS-2004-002 at http://maple.ece.utexas.edu.

[Clarke et al. ’00] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. *Counterexample- guided abstraction refinement*, In Proc. Computer-Aided Verification 2000, Lecture Notes in Computer Science. Springer, 2000.