1 Introduction

Most systems designed today have discrete components such as computer programs. Design of controllers for such systems has mostly been ad-hoc, guided by very little theory. The increasing complexity of these discrete man-made dynamic systems has rendered ad-hoc methods for their control expensive and inadequate. Some examples of such systems are manufacturing systems, computer communication networks and computer programs. We call such systems which contain discrete components as discrete event dynamical systems.

The study of control of discrete event systems was pioneered by Ramadge and Wonham[RW87b, RW89]. Since then many researchers have studied various aspects of control of discrete event systems. Stability issues have been addressed in [OWA91, BH90b, KGM93], observability issues in [CDFV88, LW88b, CM89, KGM91b], logical issues in [RW87a, KGM91a], and optimal control issues in [BH90a, KGM95, SL91]. Some of this work is summarized in the books [KG95, CL99]. Most of the previous work has focused on the *theory* of control for discrete event systems. Even though some applications such as databases, communication protocols, traffic controllers, and manufacturing control have been studied, the theory has served mostly in explaining known results rather than invent fundamentally new algorithms.

The goal of the proposed project is to extend the theory of DES to distributed systems in the context of control of distributed programs for fault-tolerance and distributed debugging. We note here that most of the work in distributed control is based on the assumption that the controllers do not communicate [LW88a, LW90, RW92, KS98, KS97, WH91, KW95, Tak98, RW95]. Control without the ability to communicate leads to very restrictive behavior. Since most distributed systems, in particular computer systems, allow some mechanism for communication, we will focus on work that allows for communication.

There has been some important work to incorporate communication in the theory of distributed control [RR98, BL98]. Ricker and Rudie[RR98] have used the modal logic for supervisory control. The use of knowledge based logic has earlier been used in analysis for distributed systems [HM84]. While this approach has been modestly successful in proving some impossibility results (e.g. achieving common knowledge is impossible in a distributed system [HM84]), it has yet to yield results in synthesis of non-trivial protocols. It is important to devise other theories in conjunction with the one based on modal logic.

The work by Barrett and LaFortune[BL98] also tackles the problem of communication. They provide necessary and sufficient conditions for control by communicating supervisory controllers (with myopic and non-myopic expectations of future communication).

More recently, Yoo and LaFortune have discussed the problem of decentralized control in [YL01a, YL01b, YL00]. They have provided an architecture for decentralized control which allows both disjunction and conjunction of the event enablement by local controllers. They also provide conditions on the plant language and the desired closed-loop language under which decentralized control is possible.

Although the work mentioned above provides many insights, it has limitations in applications to real systems. For example, they assume that there are no communication delays and that controllers are synchronized on the initial state of the system. These assumptions are not true in the present day communication networks in which communication delays play an important role in decision making. Another of their assumption is that all the communication between controllers is two-way broadcast (each message is broadcast to all controllers with the acknowledgment also as a broadcast). While this assumption may be enforced in a small system, it would lead to gross inefficiencies in medium to large scale distributed systems. The objective of the proposed project is to remove such assumptions and then apply the theory to some distributed applications.

We have in past initiated such a theory for observation and control of a distributed computation. In [GM01, MG01b], we introduce the notion of *computation slice* as an abstraction technique for predicate observation and control. Intuitively, a slice of a partial order trace (also called a computation) with respect to a predicate B is a sub-computation that contains all the global states that satisfy B. Note that the set of global states that satisfy B may be large, $O(k^n)$ in the worst case where k is the number of events in the trace of a process and n is the number of processes. Hence, one could not simply enumerate all the global states efficiently either in space or time. A slice contains all the global states that satisfy B such that it is computed efficiently (without traversing the lattice of global states) and represented concisely (without explicit representation of individual global states). A slice for a predicate is much smaller than the computation itself. Therefore, in order to detect a predicate, rather than searching the state-space of the computation, it is much more efficient to search the state-space of the slice.

As an illustration, suppose we want to detect the predicate $(x_1 * x_2 + x_3 < 5) \wedge (x_1 \geqslant 1) \wedge (x_3 \leqslant 3)$ in the computation shown in Figure 1(a). The computation consists of three processes P_1, P_2 and P_3 hosting integer variables x_1, x_2 and x_3, respectively. The events are represented by solid circles. Each event is labeled with the value of the respective variable immediately after the event is executed. For example, the value of variable x_1 immediately after executing the event c is -1. The first event on each process initializes the state of the process and every global state contains these initial events. The tail and the head of an arrow across two processes denote the send and receive of a message.

Without computation slicing, we are forced to examine all global states of the computation, twenty eight in total, to ascertain whether some global state satisfies the predicate. Alternatively, we can compute a slice of the computation with respect to the predicate $(x_1 \geqslant 1) \wedge (x_3 \leqslant 3)$ as shown in Figure 1(b). The slice is modeled by a partial order on a set of meta-events; each meta-event consists of one or more "primitive" events. A global state of the slice either contains all the events in a meta-event or none of them. Moreover, a meta-event "belongs" to a global state only if all its incoming neighbors are also contained in that global state.

The slice is modeled by a directed graph. Each vertex of the graph corresponds to a subset of events. If a vertex is contained in a global state, the interpretation is that all events corresponding to the vertex are contained in the state. Moreover, if a vertex belongs to a global state then all its incoming neighbors should also be present in that global state. We now restrict our search to the global states of the slice which are only six in number, namely $\{a, e, f, u, v\}$, $\{a, e, f, u, v, b\}$, $\{a, e, f, u, v, w\}$, $\{a, e, f, u, v, b, w\}$, $\{a, e, f, u, v, w, g\}$ and $\{a, e, f, u, v, b, w, g\}$. The slice has much fewer global states than the computation itself—exponentially smaller in many cases—resulting in substantial savings.

One can also view the slice as providing additional control to the system such that the observable set of global states do not violate the constraint $(x_1 \geqslant 1) \wedge (x_3 \leqslant 3)$. In other words, if the state visible to the external environment should satisfy $(x_1 \geqslant 1) \wedge (x_3 \leqslant 3)$, then all events in a meta-event should be executed atomically and all the precedence constraints in the slice must be satisfied.

We propose to study both algorithmic and practical issues in distributed supervisory control based on slicing and other techniques. From the algorithmic perspective, we will focus on three

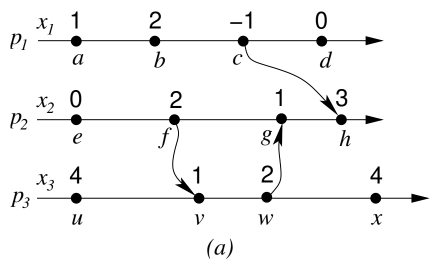

Figure 1: (a) A computation and (b) its slice with respect to $(x_1 \geqslant 1) \wedge (x_3 \leqslant 3)$.

issues. First, algorithms for slicing [GM01, MG01b] are off-line and centralized. All the traces of processes are first combined at one process and then the slice is computed. Since global predicate detection is a computation-intensive problem and multiple processors are available by the problem statement, we propose to exploit parallelism and develop distributed algorithms for slicing. Second, we will extend the technique to extend the notion of a control to controlling a system rather than a computation.

Finally, we will look at applications of supervisory control. We will look at software fault-tolerance as well as distributed debugging and testing. We first note that our approach is applicable to fault-tolerance of software systems that are under operation when the fault is detected. For many applications it is not possible to find the defect in the program that caused the fault and run the program from the beginning. For example, while operating an airplane (or a shuttle) it is not a viable option for the program to be modified and then re-executed. One needs to control the computation such that the software fault is by-passed. This course of action is indeed possible when it is a synchronization fault and the controller can avoid the fault by imposing additional order between events. Of course, all the faults that are detected can be logged and debugged for later.

Many of the problems in distributed debugging can also be cast into the framework of observation and control [Gar97]. While issues in observation of a distributed program has been studied by various researchers [CM91a, GW94, GW96, HMRS98], the issue of control has received less attention [Gar97]. The abstraction of automatic control provides great power to the user while testing a distributed program. Testing, in essence, is observing the behavior of a distributed program under control.

Besides investigating theoretical issues and algorithms, we also propose to develop an experimental testing system that incorporates our algorithms. We will use Java for building our prototype which will be targeted towards testing Java applications, especially replicated distributed servers. We will support testing of distributed programs written using either Java sockets or remote method invocations.

We note here that the proposed project requires expertise in supervisory control of discrete event systems as well as distributed systems. We have worked extensively in both of these areas and one of the goals of the project is cross-fertilization of ideas from one discipline to the other.

This proposal is organized as follows. Section 2 describes our prior work supported by NSF. The remaining sections describe the proposed project. Section 3 describes the concept of slicing a distributed computation that forms the basic technique for the project. Section 4 describes our proposed work in the theory and applications of distributed control. Finally, Section 5 discusses our plan and broader impacts of work.

2 Prior NSF Results

NSF Award Number: ECS-9907213. Title of the Project: "Distributed Supervisory Control: Theory and Applications," Amount: $224000, Period of Support: October 1999 - August 2003.

2.1 Summary of the Results

In this section, we describe some of our earlier work sponsored by NSF. Much of our work from the prior NSF support pertains to observation and control of a distributed program.

2.1.1 Observation of Global Predicates

As mentioned earlier, monitoring of global predicates is a fundamental problem in asynchronous distributed systems that arises in various contexts such as design, testing and debugging, and fault-tolerance of distributed programs. It was known from earlier results that detecting a global predicate in 3-CNF form is NP-complete but a simple conjunction of local predicates (1-CNF form) can be detected efficiently. The complexity of detecting a predicate in 2-CNF was previously open. In [MG01a], we close the wide gap between intractability and efficient computation of global predicates. we establish that the problem of determining whether there exists a global state of a computation that satisfies a predicate in k-CNF, $k > 1$, in which no two clauses contain variables from the same process is NP-complete in general. A polynomial-time algorithm to find the global state, if it exists, that satisfies the predicate for special cases is provided.

Furthermore, we present an algorithm to determine whether there exists a global state of a computation for which the sum $x_1 + x_2 + ... + x_n$ exactly equals some constant k, where each x_i is an integer variable on process p_i such that it is incremented or decremented by at most one at each step. As a corollary, any symmetric global predicate on boolean variables such as absence of simple majority and exclusive-or of local predicates can be detected. Additionally, the problem is proved to be NP-complete if each x_i can be changed by an arbitrary amount at each step.

In [SG02], we initiated the study of predicate detection algorithms for simple *unnested* temporal logic predicates. We use the temporal logic, CTL, for specifying properties of a distributed computation and interpret it on a finite lattice of global states. The standard interpretation structures of CTL are trees with infinite sequences of global states reflecting the behavior of reactive and concurrent systems which may be continuously alive. We make use of the lattice structure of the global states in developing algorithms for predicate detection. We present solutions to the predicate detection of linear and observer-independent predicates under EG and AG operators of CTL. Linear predicates are those that are closed under meet operation and observer-independent predicates are those for which EF and AF modalities are equivalent. For linear predicates we develop polynomial-time predicate detection algorithms which exploit the structure of finite distributive lattices. For observer-independent predicates we prove that predicate detection is NP-complete under EG operator and co-NP-complete under AG operator. We also present polynomial-time algorithms for a CTL operator called *until*, for which such algorithms did not exist. This operator helps in detecting properties where a condition has to hold until another condition eventually holds. Finally, this work unifies many earlier results in predicate detection in a single framework.

2.1.2 Control of Global Predicates

In [TG01], we discuss the predicate control problem useful for software fault-tolerance. Concurrent programs often encounter failures, such as races, owing to the presence of synchronization faults (bugs). The growing prevalence of concurrent programs and the need to maintain availability in spite of failures make the problem of tolerating synchronization faults increasingly important. One existing technique to tolerate synchronization faults is to roll back the program to a previous state and re-execute. Since synchronization failures are usually transient in nature, they may not occur in the re-execution. Instead of relying on chance, our approach is to control the re-execution in an attempt to avoid a recurrence of the synchronization failure. The control is achieved by tracing information during an execution and using this information to add synchronizations during the re-execution.

This approach gives rise to the predicate control problem, which takes a computation and a property specified on the computation, and outputs a controlled computation that maintains the property. We solve the predicate control problem for the mutual exclusion property, which is important in synchronization fault tolerance. Our solutions include simple mutual exclusion, as well as read-write mutual exclusion, independent mutual exclusion, and independent read-write mutual exclusion. We demonstrate how the techniques can be applied to tolerate synchronization faults. Furthermore, the techniques have other application domains such as concurrent debuggers and process control systems.

In [TG01], it is also shown that the predicate control problem is NP-complete in general. In [MG00], we define a class of global predicates, called region predicates, that can be controlled efficiently in a distributed computation. We prove that the synchronization generated by our algorithm is optimal. Further, we introduce the notion of an admissible sequence of events and prove that it is equivalent to the notion of predicate control. We then give an efficient algorithm for the class of disjunctive predicates based on the notion on an admissible sequence.

2.2 Algebraic Techniques for Stochastic Systems

In [GKM99], we have introduced probabilistic languages for modeling the qualitative behavior of stochastic discrete event systems. A probabilistic language is a unit interval valued map over the set of traces of the system satisfying certain consistency constraints. We have shown that regular language operators such as choice, concatenation, and Kleene-closure can be defined in the setting of probabilistic languages to allow modeling of complex systems in terms of simpler ones. Further, the set of probabilistic languages is closed under such operators thus forming an algebra. It also is a complete partial order under a natural ordering in which the operators are continuous. Hence recursive equations can be solved in this algebra. This is alternatively derived by using contraction mapping theorem on the set of probabilistic languages which is shown to be a complete metric space. The notion of regularity, i.e., finiteness of automata representation of probabilistic languages has been defined and shown that regularity is preserved under choice, concatenation, and Kleene-closure. We show that this formalism is also useful in describing system performances such as completion time, reliability, etc. and present properties to aide their computation.

In [KG], we have studied supervisory control problems for stochastic systems. As before, the event set is partitioned into the set of controllable and uncontrollable events. The supervisor can disable any of the controllable events. When the controller disables some events, the probability of enabled events is normalized using the probability of disabled events, The control objective may be specified as a pair of probabilistic languages which provide the lower and the upper bound on the desired behavior. Our framework generalizes that of Ramadge and Wonham which corresponds to the special case when all languages are deterministic.

2.2.1 Computation Slicing

Computation slicing was discovered as part of this project and reported in [GM01] and [MG01b]. We do not discuss these papers as they are described in Section 3.

2.3 Broader Impact of prior NSF work

Prior NSF support has also helped in development of academic infrastructure. In particular, I have finished a graduate-level textbook on asynchronous distributed computing [Gar02] published by Wiley & Sons. The theme of observation and control is central in the book and it incorporates many new research ideas that had appeared only in conferences and journal.

The NSF funding also helped in education of graduate students in our department. Neeraj Mittal, Alper Sen, James Roller Jr. and Ashis Tarafdar were supported by the funding for part of their Ph.D. duration.

2.4 Selected List of Publications Acknowledging the NSF Award

- R. Kumar, V. K. Garg, "Control of Stochastic Discrete Event Systems Modeled by Probabilistic Languages," *IEEE Transactions on Automatic Control*, Volume 46, No. 1, April 2001, pp. 593 – 606.

- O. Damani, Yi-Min Wang and V. K. Garg, "K-Optmistic Message Logging," *Journal on Parallel and Distributed Computing*, accepted 2003.

- R. Kumar, V. K. Garg, "Computation of State Avoidance Control for Infinite State Systems in Assignment Program Framework," *IEEE Transactions on Robotics and Automation*, accepted 2003.

- A. Tarafdar, V. K. Garg, "Predicate Control," *Journal of Parallel and Distributed Computing*, accepted 2003.

- V. K. Garg, R. Kumar, S. I. Marcus, "A Probabilistic Language Formalism for Stochastic Discrete Event Systems," *IEEE Transactions on Automatic Control*, Vol. 44, No. 2, February 1999, pp. 280 – 293.

- Neeraj Mittal and Vijay K. Garg, An Experimental Study on Computation Slicing, *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, Baltimore, Maryland, 2003, pp. 105 – 113.

- Vijay K. Garg, Algorithmic Combinatorics based on Slicing Posets, *Proc. 22nd Conference on the Foundations of Software Technology & Theoretical Computer Science (FSTTCS)*, Kanpur, India, December 2002, pp. 169 – 181.

- Vijay K. Garg, C. Skawratananond, On Timestamping Synchronous COmputations, *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, July 2002, pp. 552–560.

- Alper Sen and Vijay K. Garg, Detecting Temporal Logic Predicates in the Happened Before Model, International Parallel and Distributed Processing Symposium (IPDPS'02). April, Florida, pp. 76.

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

- Vijay K. Garg, C. Skawratananond, String Realizers of Posets with Applications to Distributed Computing, ACM Symposium on Principles of Distributed Computing (PODC'01), August 26-29, 2001, Newport, Rhode Island, pp. 72 – 80.

- Neeraj Mittal and Vijay K. Garg, Efficient Algorithms for Detecting Global Properties in Distributed Computations, *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'01)*, Phoenix, May 2001, pp. 3 – 10.

- Vijay K. Garg and Neeraj Mittal, On Slicing a Distributed Computation, *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'01)*, Phoenix, May 2001 (nominated for the best paper award), pp. 322 – 329 .

- Neeraj Mittal, Vijay K. Garg, Debugging Distributed Programs Using Controlled Re-execution, ACM Symposium on Principles of Distributed Computing (PODC'00), Portland, Oregon, July 2000, pp. 239 – 248.

3 Slicing

We first give the mathematical basis of computation slicing. The concept of slicing exploits the fact that the set of all global states of a computation forms a distributive lattice. It then uses Birkhoff's Representation Theorem for Finite Distributive Lattices [DP90] which we describe next.

Given a lattice, its meet (infimum) and join (supremum) operators are denoted by $\sqcap$ and $\sqcup$, respectively. A lattice is *distributive* if meet distributes over join [DP90]. It can be proved that meet distributes over join iff join distributes over meet. An element of a lattice is *join-irreducible* if it cannot be expressed as join of two distinct elements (of the lattice), both different from itself [DP90].

Let L be a lattice and $\mathcal{J}(L)$ be the set of its join-irreducible elements. In case L is a distributive lattice, it satisfies an important property. Specifically, every element in L can be expressed as join of some subset of elements in $\mathcal{J}(L)$ and vice versa [DP90, Birkhoff's Theorem]. In other words, $\mathcal{J}(L)$ completely characterizes L. This is significant because $|\mathcal{J}(L)|$ is generally much smaller—exponentially in many cases—than $|L|$. Hence if some computation on L can instead be performed on $\mathcal{J}(L)$, we obtain a significant computational advantage.

Consider a computation $\langle E, \rightarrow \rangle$ where E is the set of events in the computation and $\rightarrow$ is the Lamport's happen-before relation on the set [Lam78]. Let $\mathcal{C}(E)$ denote the set of its global states.

$\mathcal{C}(E)$ forms a *distributive* lattice under the relation $\subseteq$; its join and meet operators correspond to set union ($\cup$) and set intersection ($\cap$), respectively.

Now, consider a subset $\mathcal{D} \subseteq \mathcal{C}(E)$. $\mathcal{D}$ forms a *sublattice* of $\mathcal{C}(E)$ if $\mathcal{D}$ is closed under set union and set intersection. That is, given two global states from $\mathcal{D}$, the global states obtained by their set union and set intersection also belong to $\mathcal{D}$. It can be proved that any sublattice of a distributive lattice is also a distributive lattice [DP90]. Thus if $\mathcal{D}$ is a sublattice of $\mathcal{C}(E)$, then, using Birkhoff's Theorem, $\mathcal{J}(\mathcal{D})$ completely characterizes $\mathcal{D}$. This forms the basis for the notion of computation slice.

The slice of a computation with respect to a predicate is the directed graph satisfying the following two conditions. First, it contains all global states for which the predicate evaluates to true. Second, among all computations that fulfill the first condition, it contains the least number of global states. Intuitively, slice is a *concise representation* of global states satisfying certain property.

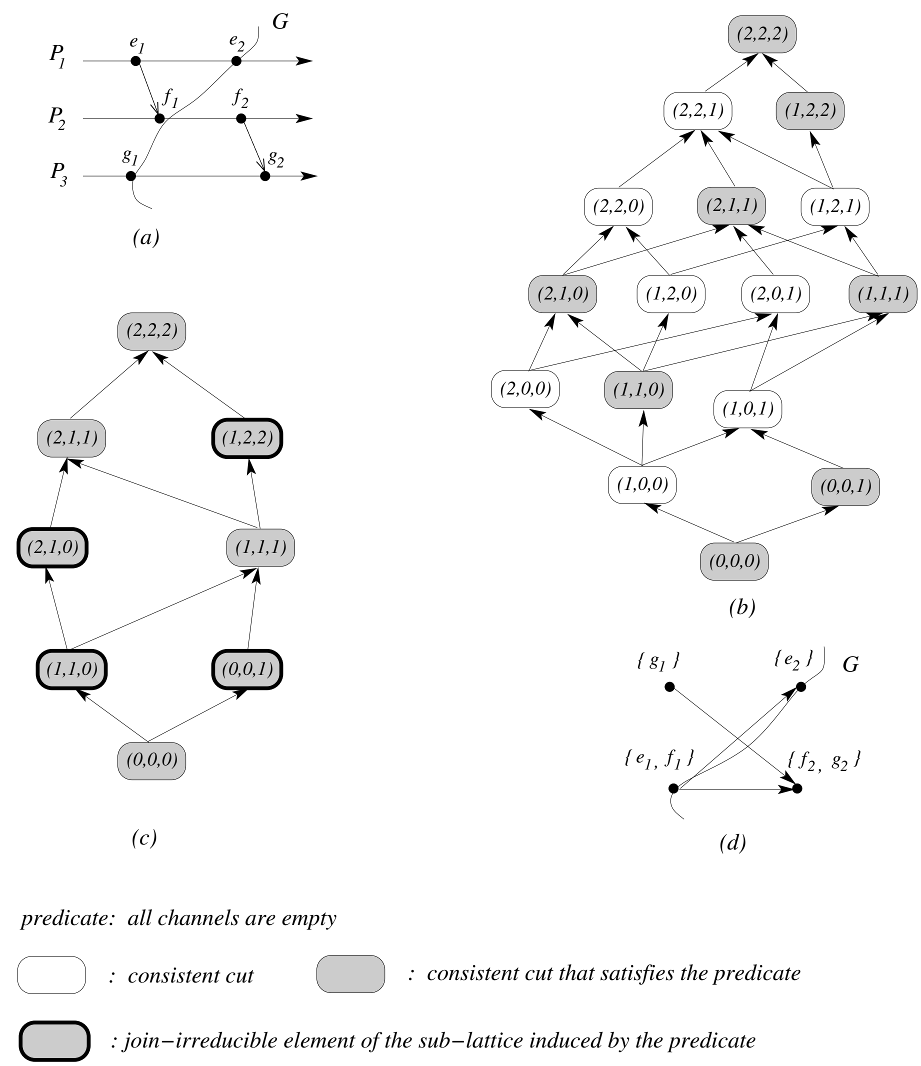

Figure 2: A computation (a), its lattice (b), a sub-lattice (c) and the corresponding slice (d).

As an example of a slice, consider the computation in Figure 2(a). The lattice corresponding to the computation is show in Figure 2(b). The label of each element in the lattice is a 3-tuple denoting the index of the maximal event on each process contained in the corresponding global state. For example, the label for the global state G is $(2, 1, 1)$. For the predicate "all channels are empty", the sublattice is shown in Figure 2(c) along with its join-irreducible elements. The slice of the computation with respect to the predicate is shown in Figure 2(d). It can be verified the set of global states of the slice are exactly those that satisfy the given predicate in the original computation. In the slice, as expected, events e_1 and f_1 and events f_2 and g_2 must be executed atomically.

We show in [MG01b] that:

Theorem 1 *The slice exists and is uniquely defined for all predicates.*

In general, a slice may also contain global states that do not satisfy the predicate. In case a slice does not contain any such cut, it is called a *lean* slice [MG01b]. We have identified a class of *regular predicates* for which the slice is *lean*. In other words, the slice for a regular predicate contains precisely those global states for which the predicate evaluates to true. A predicate is regular if the set of global states that satisfy a regular predicate is closed under set union and set intersection. Some examples of regular predicates are: conjunction of local predicates like "all processes are in red state", and channel predicates such as "all control messages have been received". We proved that the class of regular predicates is closed under conjunction, that is, the conjunction of two regular predicates is also a regular predicate.

We have devised an efficient algorithm to compute the slice for a regular predicate. The time-complexity of the algorithm is $O(n^2|E|)$, where n is the number of processes and E is the set of events. Additionally, for special cases of regular predicates such as conjunction of local predicates and certain channel predicates, we have developed *optimal* algorithms for computing the slice which have $O(|E|)$ time-complexity.

We have employed slicing for predicate detection in a distributed computation. A predicate can be detected under four modalities, namely *possibly*, *definitely*, *invariant* and *controllable*

[CM91b, GW91, TG99, MG00]. A predicate is *possibly* true in a computation if there is a global state of the computation that satisfies the predicate. On the other hand, a predicate *definitely* holds in a computation if it is true in some global state in all paths from the initial global state to the final global state in the lattice of global states. The former modality is useful for expressing the violation of a safety property, whereas the latter can be used to specify liveness properties. The modalities *invariant* and *controllable* are duals of the modalities *possibly* and *definitely*, respectively. We present the following polynomial-time predicate detection algorithms in [MG01b]:

Theorem 2 *A regular predicate is* **possibly** *true in a computation if and only if the slice of the computation with respect to the predicate is non-empty. It is* **invariant** *in a computation if and only if the slice of the computation with respect to the predicate is same as the computation. It is* **controllable** *in a computation if and only if the slice of the computation with respect to the predicate has the same number of strongly connected components as the computation.*

We have developed a *composition* technique for combining two slices. Specifically, given two slices, composition involves computing either (1) the smallest slice that contains all global states common to both the slices, or (2) the smallest slice that contains all global states that belong to at least one of the slices. The former corresponds to composition with respect to conjunction the latter to composition with respect to disjunction. We have applied composition to compute the slice for the complement of a regular predicate, referred to as *co-regular predicate*. In particular, in [MG01b], we show that:

Theorem 3 *Given a computation, a co-regular predicate can be expressed as disjunction of* $O(n|E|)$ *number of regular predicates.*

The slice for a co-regular predicate can, therefore, be computed by composing the slices for regular predicates corresponding to the disjuncts. Furthermore, we have also used composition to compute the slice for a k-local predicate for constant k. Our algorithm uses the following result by Stoller and Schneider [SS95]: given a computation, a k-local predicate, for constant k, can be written as disjunction of polynomial number of regular predicates.

We show in [MG01b] that it is computationally hard in general to compute the slice for a predicate. Nonetheless, it is still useful to be able to compute an *approximate slice* for such a predicate efficiently. An approximate slice may be bigger than the actual slice but will be much smaller than the computation itself. Using composition, it is possible to compute an approximate slice—in polynomial-time—for a predicate composed from linear predicates, co-regular predicates and k-local predicates, for constant k, using $\wedge$ and $\vee$ operators. Example of such a predicate is: $(x_1 \vee \neg x_2) \wedge (x_3 \vee \neg x_1) \wedge (x_2 \vee x_3)$, where each x_i is a regular predicate.

3.1 Experimental Evaluation

We implemented our slicing algorithms in Java in a prototype tool called Partial Order Trace Analyzer (POTA) [SG03]. The overall structure of POTA architecture is shown in Figure 3. The tool consists of 3 main modules; analyzer, translator, and instrumenting module. We have implemented slicing based predicate detection algorithms in the analyzer module. The translator module translates traces (slices) into Promela [Hol97] (input language of SPIN model checker) and serves the following purposes: First, it enables us to compare POTA with SPIN for the same input partial order execution trace. Second, when the slice is not lean it may contain states that do not satisfy the predicate and we may need to search the state space of the slice. The translator module enables using SPIN to conduct this search. Note that SPIN contains implementation of various reduction techniques including partial-order reduction. The instrumentation module generates an instrumented program such that when run every process outputs its local state where a local state contains the values of variables relevant to the predicate in question and a vector clock that is updated for each internal, send and receive event according to the Fidge/Mattern [Fid91, Mat89] algorithm. Vector clocks enable us to obtain a partial order representation of traces. Currently, the instrumentation is being done manually for message passing programs. However, we can also generate a partial order trace for programs with shared variables using partial order generation techniques described in [Chr02, SRA03].

In order to evaluate the effectiveness of POTA, we performed experiments with scalable protocols such as distributed dining philosophers, General Inter-ORB Protocol (GIOP), and leader election. All experiments were performed on a 1.4 Ghz Pentium 4 machine running Linux. We restricted the memory usage to 512 MB, but did not set a time limit. The two performance metrics we measured are running time and memory usage. In the case of slicing both metrics also include

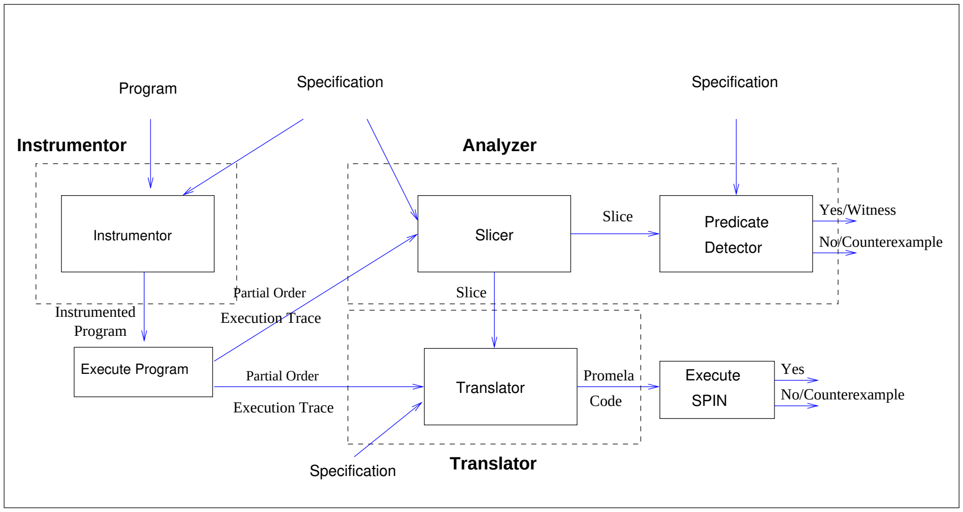

Figure 3: Overview of POTA Architecture

the overhead of computing the slice. We run the programs for 20 seconds and our measurements are averaged over 20 traces for each program. The detailed results of the experiments of this section are available from [Par].

The first protocol implements dining philosophers protocol in Java [Har98]. The complement of the mutual exclusion property corresponds to possibly: $\bigvee_{i,j \in 0 \ldots (n-1)} \left(eat_i \wedge eat_j \right)$ and denotes that two neighboring philosophers i and j do not eat at the same time. Figure 4 displays our comparison results. It took SPIN 250s and 300MB to exhaustively verify the property for six processes and it ran out of memory for seven processes. Whereas with POTA we can handle 200 processes under 400s and 60MB. In fact, the 200 process limit is due to the fact that we could not generate traces with more than 200 processes since JVM ran out of memory.

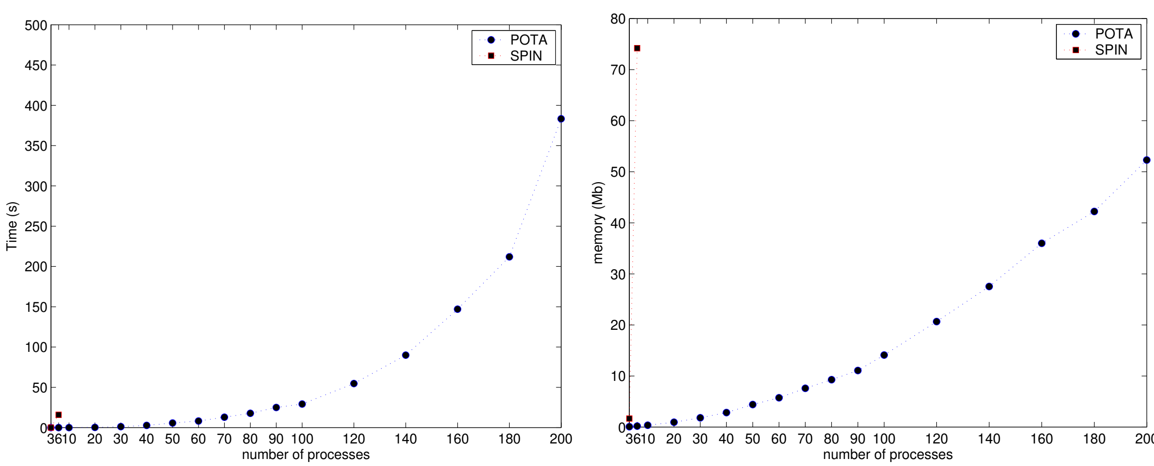

Figure 4: Dining Philosophers verification results. SPIN runs out of memory for $n > 6$

The second protocol GIOP is a central feature of the Common Object Request Broker Architecture (CORBA) that aids in achieving the desired interoperability between ORBs. The CORBA specification defines GIOP as a standard protocol to allow communication of object invocations between ORBs. One invariant of the protocol states that "If the user received system exception then its request was processed at most once".

invariant: $\Big(\neg SystemException_i \vee \Big(\bigvee_k \bigwedge_j Server_j Processed_i = m\Big) \vee \Big(\bigwedge_l Server_l Processed_i = 0\Big)\Big)$, where $m = 1$ if $k = j$ and $m = 0$ otherwise, for all users i and for all servers j, k, l. Note that for this predicate the slice may be approximate.

Kamel and Leue [KL00] could not complete exhaustive verification of GIOP for a configuration with 10 processes. Instead, they verified a simplified version of the protocol without server migration functionality. Even with an execution trace input, SPIN failed to complete verification with more than 10 processes using 512 MB. In one case, we generated execution traces of unsimplified GIOP protocol for a configuration with 120 processes and completed verification with POTA. In another scenario, we injected faults into the protocol and analyzed the resulting execution traces. SPIN again failed to find the faults before running out of memory for configurations with more than 10 processes. However, POTA was able to find the faults easily.

Third, we test the leader election program that implements the Chang-Roberts algorithm where processes are arranged in a unidirectional ring. The liveness property in this case requires that eventually all processes know that election is *done* and at least one process has its $leaderFlag$ variable set to true, that is, controllable: $\Big(\bigvee_i \neg done_i \vee \Big(\bigwedge_j \neg leaderFlag_j\Big)\Big)$. Note that this predicate contains disjunction operator and the slice may be approximate. The approximate slice was the same as the computation in almost all cases. Therefore, slicing did not fair well for leader election protocol. Note that the approximate slice was much smaller than the computation in the case of GIOP verification above. Both SPIN and POTA ran out of memory for 12 processes. In fact, SPIN performed better than POTA due to the computation of slices. Slice computation took at most 4 MB and 3 seconds.

In almost all cases, our algorithms are significantly faster and space efficient than SPIN. We have measured over three orders of magnitude gain over SPIN in some experiments. We note here that SPIN is a tool for checking a temporal logic formula for a program whereas POTA is a tool for checking a temporal logic formula for a partial order. Therefore, the comparison between these tools does not imply that POTA is better than SPIN. SPIN is a more general tool. We are using SPIN as a comparison tool because we do not know of any other tool that checks a temporal logic formula on a partial order.

4 Proposed Work

We propose to work on theory, implementation and applications of distributed supervisory control as described next.

4.1 Efficient Algorithms for Observation of nested Temporal Logic Predicates

So far our focus has been on global predicates whose truth value depend on a single global state. Many specifications of distributed programs, however, are temporal in nature because we are interested in properties related to the sequence of events during an execution. For example, the liveness property in dining philosophers problem, "a philosopher, whenever gets hungry, eventually gets to eat", is a temporal property.

We use the temporal logic CTL [EE81] to write specifications of distributed programs. Basic temporal operators of this logic include **EF**, **AF**, **EG**, and **AG**. We say that temporal predicate $\mathbf{EF}(p)$ (resp. $\mathbf{EG}(p)$) holds at a global state G if and only if for some path starting from G, the predicate p eventually (resp. always) holds on the path. We say that temporal predicate $\mathbf{AF}(p)$ (resp. $\mathbf{AG}(p)$) holds at a global state G if and only if for all paths starting from G, the predicate p eventually (resp. always) holds on all the paths. Figure 6 illustrates these temporal predicates on a lattice of global states.

We plan to develop algorithms to detect nested temporal predicates expressed in CTL. The notion of slicing is crucial for design of these algorithms. For example, to check the temporal predicate $\mathbf{AG}(hungry_i \Rightarrow \mathbf{AF}(eating_i))$, we need to verify that for every state in which $hungry_i$ is true, all paths starting from that state lead to a state in which $eating_i$ is true. Given an algorithm to compute slice for $\mathbf{AF}(p)$, we can compute the set of global states in which $\neg hungry_i \vee \mathbf{AF}(eating_i)$ holds. This set should include all global states.

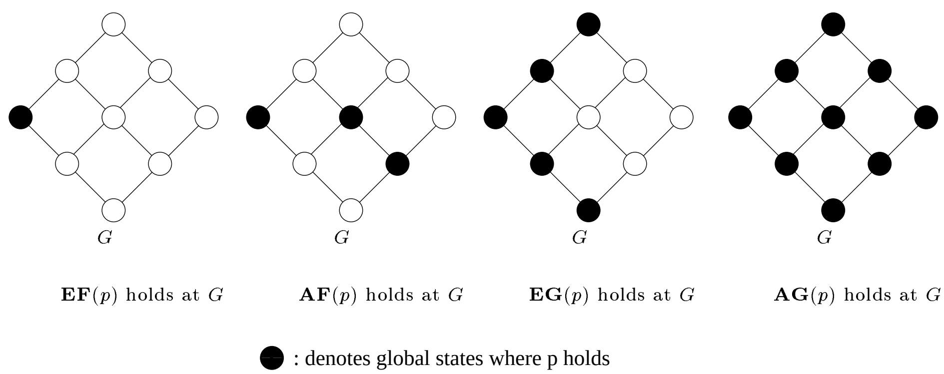

Figure 5: Basic CTL operators

4.2 Controlling a set of computations

So far our control has been restricted to the order of events executed at different processes. The traditional Ramadge-Wonham model uses control based on disabling and enabling controllable events. If a computation cannot be controlled by changing the order of events, it may still be controllable by disabling certain events so that the system is steered towards a different computation. Note that this requires control of a system rather than the control of a computation as investigated by our earlier work. Since a distributed system can be viewed as a set of posets/computations, the initial goal would be to extend our technique for determining the control with respect to multiple posets rather than a single poset. As before our emphasis would be on polynomial time and polynomial space algorithms for control. In particular, our theory would avoid the construction of the set of all global states.

One can also look at this problem from the perspective of reachability graph of a distributed system with finite state processes and channels of finite capacity. The reachability graph of such a system is finite, but generally very large (exponential in the number of processes). The goal of the research is to derive efficient algorithms to determine a small number of distributive lattices that covers the entire reachability graph. If we succeed in doing this, one would have efficient algorithms based on join-irreducibles of the lattices rather than global states of the graph.

4.3 Alleviating State Space Explosion Using Congruences

One of the major bottlenecks in controlling and analyzing distributed systems is the problem of combinatorial state explosion. As it is very well-established in literature and practice, one can alleviate the state explosion problem by using equivalence classes on state space. For our purposes, this corresponds to analyzing quotient lattices of the global state graph. The problem can be posed as follows: Given a temporal logic formula that needs to be maintained (controlled), how can one define the congruence such that the formula is valid on the quotient lattice iff it is valid in the general lattice. Such questions have been investigated in the context of general graphs. Our goal is to investigate these questions for distributive lattices where the algorithms would use only the join-irreducible elements of the lattice for the analysis of controllability of the formula.

4.4 Distributed Algorithms for Decentralized Control

By computing a slice of the computation with respect to a predicate, one can determine the control that needs to be exercised for maintaining the predicate. However, our algorithm for computing this control is currently centralized. Once the control is determined it is relayed to various processes that can exercise the control. In this project, we will investigate techniques in which processes determine local control based on partial information at other sites. There are many issues related with this technique. How much information should processes relay to each other so that optimal control can be determined? If the observation map is fixed, can the predicate be controlled with respect to that observation map?

4.5 Application: Distributed Debugging and Testing

The problem of control arises in multiple ways during distributed debugging. The ability to control the message ordering is useful for the programmer to test the program with different message and event ordering. As a simple example, assume that the user is interested in testing a distributed program when one of the servers is always in the red state (the red state could signify any application specific local condition such as "busy" or "waiting"). To ensure this manually is difficult, if not impossible. As another example, assume that the user suspects that a bug may arise if *red* messages are always overtaken by *green* messages. Again testing the program in this scenario manually is difficult.

If a distributed program is interacting with an external observer, the relative speed of different processes may affect the outcome. In the proposed method, the system will automatically control the speed of various processes so that the desired predicate becomes true. For example, in distributed debugging, a programmer is usually interested in determining if a global state that violates any safety property is reachable.

In distributed debugging, the concept of unobservable and uncontrollable events arises very naturally. A process may not be able to observe the current state of other processes. Similarly, failure of a process, or input from an external process may be uncontrollable.

We note here that even though there are many testing techniques [TC96], there is little work on control of distributed executions as proposed in this project. The work by Damodaran-Kamal and Francioni [DKF93] discusses controlled execution. However, the problem they study is different and their method of testing has an exponential time complexity. Many papers on testing of distributed programs deal with race detection [NBDK96, CL95]. The other technique for testing fault-tolerant distributed programs include failure injection as in [AC97, DJMT96]. These are important techniques for debugging and would be used in conjunction with those proposed in this project.

4.6 Application: Software Fault-Tolerance

We also propose to investigate software fault-tolerance as an application of this project. Under this model all software runs in a supervised lookahead mode. If the controller finds that the lookahead computation can violate the safety predicate, it controls the system so that the safety predicate remains invariant. Recently, the principal investigator has finished a book on distributed programming in Java (published by Wiley & Son). As part of the book, the investigator has a collection of distributed Java programs for many applications. These applications include algorithms for handling distributed data, managing distributed resources, detecting global properties etc. Currently, these programs are run in unsupervised mode and can result in unsafe state when the program has a bug or some fault occurs during execution. Once our system is ready we will be able to run this suite of programs in supervised mode which can tolerate a large class of faults.

5 Plan of work and Broader impact of work

The proposed project will be carried out in three phases - each of one year duration. In the first phase, we will focus mainly on developing the theory for distributed control. In the second phase, we will also develop incremental and distributed on-the-fly algorithms for constructing slices. In the third phase, we will incorporate our algorithms in POTA and extend temporal logic predicate detection algorithms. In this phase, we will also evaluate effectiveness of our approach by using POTA system for automatically detecting faults and controlling the computation to bypass the faults for a wide set of distributed programs.

The project will result in theoretical and practical advances in supervisory control of discrete event systems. With these advances, it is expected that all highly dependable systems will always use supervisory control to avoid catastrophic faults. This project will also have impact on the development of human resources. Two graduate students will be trained in the fundamentals of discrete event systems and distributed systems.

References

[AC97] G. A. Alvarez and F. Cristian. Centralized failure injection for distributed fault-tolerant protocol testing. In *Proc. of the IEEE International Conference on Distributed Computing Systems*, pages 78–85, June 1997.

[BH90a] Y. Brave and M. Heymann. On optimal attraction in discrete event processes. Technical Report CIS 9010, Technion - Israel Institute of Technology, Hafia, Israel 32000, 1990.

[BH90b] Y. Brave and M. Heymann. "On Stabilization of Discrete Event Processes". *Internation Journal Control*, 51:1101–1117, 1990.

[BL98] G. Barret and S. Lafortune. Decentralized supervisory control with communication. Technical Report CGR-98-12, University of Michigan, 1998.

[CDFV88] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete event processes with partial observation. *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988.

[Chr02] M. Christiaens. Dynamic techniques for the optimization of data race detection. In *Proc. of Program Acceleration through Application and Architecture driven Code Transformation Symposium*, pages 73–75, September 2002.

[CL95] R. Cypher and E. Leu. Efficient race detection for message-passing programs with nonblocking sends and receives. In *Proceedings of the Symposium on Parallel and Distributed Processing*, pages 534 – 541, May 1995.

[CL99] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, Boston, MA, 1999.

[CM89] H. Cho and S. I. Marcus. On supremal languages of class of sublanguages that arise in supervisor synthesis problems with partial observations. *Mathematics of Control Signals and Systems*, 2:47–69, 1989.

[CM91a] 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.

[CM91b] R. Cooper and K. Marzullo. Consistent Detection of Global Predicates. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, California, 1991.

[DJMT96] S. Dawson, F. Jahanian, T. Mitton, and T. Tung. Testing of fault-tolerant and real-time distributed systems via protocol fault injection. In *Proc. of the International Symposium on Fault-Tolerant Computing*, pages 404 – 414, June 1996.

[DKF93] S. K. Damodaran-Kamal and J. Francioni. Nondeterminacy: Testing and debugging message passing parallel programs. In Barton P. Miller and Charles McDowell, editors, *Proceedings of the Workshop on Parallel and Distributed Debugging*, volume 28-12 of *ACM SIGPLAN Notices*, pages 118–128, New York, NY, May 1993. ACM Press.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, UK, 1990.

[EE81] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In *Proc. of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, Yorktown Heights, New York, May 1981.

[Fid91] C. Fidge. Logical Time in Distributed Computing Systems. *IEEE Computer*, 24(8):28–33, August 1991.

[Gar97] V. K. Garg. Observation and control for debugging distributed computations. In *Proc. Third International Workshop on Automated Debugging (AADEBUG'97)*, pages 1–12, May 1997. Keynote presentation.

[Gar02] V. K. Garg. *Elements of Distributed Computing.* Wiley & Sons, New York, NY, 2002.

[GKM99] V. K. Garg, R. Kumar, and S.I. Marcus. A probabilistic language formalism for stochastic discrete event system. *IEEE Transactions on Automatic Control*, 44(2):280 – 293, 1999.

[GM01] V. K. Garg and N. Mittal. On slicing a distributed computation. In *21st International Conference on Distributed Computing Systems (ICDCS' 01)*, pages 322–329, Washington - Brussels - Tokyo, April 2001. IEEE.

[GW91] V. K. Garg and B. Waldecker. Detection of Unstable Predicates. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*, Santa Cruz, California, May 1991.

[GW94] 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.

[GW96] V. K. Garg and B. Waldecker. Detection of strong unstable predicates in distributed programs. *IEEE Transactions on Parallel and Distributed Systems*, 7(12):1323–1333, December 1996.

[Har98] S. Hartley. *Concurrent Programming: The Java Programming Language.* Oxford University Press, 1998.

[HM84] J.Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *Proc. of the ACM Symposium on Principles of Distributed Computing*, pages 50 – 61, 1984.

[HMRS98] Michael Hurfin, Masaaki Mizuno, Michel Raynal, and Mukesh Singhal. Efficient distributed detection of conjunctions of local predicates. *IEEE Transactions on Software Engineering*, 24(8):664–677, August 1998.

[Hol97] G. J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5), May 1997.

[KG] R. Kumar and V. K. Garg. Control of stochastic discrete event system modeled using probabilistic languages. *IEEE Transactions on Automatic Control*. submitted.

[KG95] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publisher, Norwell Massachusetts, 1995.

[KGM91a] R. Kumar, V. K. Garg, and S. I. Marcus. Using predicate transformers for supervisory control. In *Proceedings of 1991 IEEE Conference on Decision and Control*, pages 98–103, Brighton, UK, December 1991.

[KGM91b] R. Kumar, V.K. Garg, and S.I. Marcus. "On Controllability and Normality of Discrete Event Dynamical Systems". *Systems and Control Letters*, 17:1157–1168, 1991.

[KGM93] R. Kumar, V. K. Garg, and S. I. Marcus. Language stability and stabilizability of discrete event dynamical systems. *SIAM Journal of Control and Optimization*, 31(5):1294–1320, September 1993.

[KGM95] R. Kumar, V.K. Garg, and S.I. Marcus. Optimal supervisory control of discrete event systems. *SIAM Journal on Control and Optimization*, 33(2):419–439, 1995.

[KL00] M. Kamel and S. Leue. Formalization and Validation of the General Inter-ORB Protocol (GIOP) Using Promela and SPIN. *Software Tools for Technology Transfer*, 2(4):394–409, April 2000.

[KS97] R. Kumar and M. A. Shayman. Centralized and decentralized supervisory control of nondeterministic systems under partial observation. *SIAM Journal of Control and Optimization*, 35(2):363–383, March 1997.

[KS98] R. Kumar and M. A. Shayman. Formulae relating controllability, observability, and co-observability. *Automatica*, 34(2):211–215, 1998.

[KW95] P. Kozak and W. M. Wonham. Fully decentralized solutions of supervisory control problems. *IEEE Transactions on Automatic Control*, 40(12):2094–2097, 1995.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[LW88a] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete event systems. *Information Sciences*, 44:199–224, 1988.

[LW88b] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[LW90] F. Lin and W. M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions of Automatic Control*, 35(12):1330–1337, December 1990.

[Mat89] F. Mattern. Virtual Time and Global States of Distributed Systems. In *Parallel and Distributed Algorithms: Proceedings of the Workshop on Distributed Algorithms (WDAG)*, pages 215–226. Elsevier Science Publishers B. V. (North-Holland), 1989.

[MG00] N. Mittal and V. K. Garg. Debugging distributed programs using controlled re-execution. In *19th Annual ACM Symposium on Principles of Distributed Computing (PODC-00)*, pages 239 – 248. ACM, July 2000.

[MG01a] N. Mittal and V. K. Garg. On detecting global predicates in distributed computations. In *21st International Conference on Distributed Computing Systems (ICDCS' 01)*, pages 3–10, Washington - Brussels - Tokyo, April 2001. IEEE.

[MG01b] N. Mittal and V. K. Garg. Slicing a distributed computation: Techniques and theory. In *5th International Symposium on DIStributed Computing (DISC'01)*, pages 78 – 92, October 2001.

[NBDK96] R. H. B. Netzer, T. W. Brennan, and S. K. Damodaran-Kamal. Debugging race conditions in message-passing programs. In *Proceedings of the Symposium on Parallel and Distributed Tools*, May 1996.

[OWA91] C.M. Ozveren, A.S. Willsky, and P.J. Antsaklis. "Stability and stabilizability of discrete event dynamic systems". *J. Association for Computing Machinery*, 38(3), July 1991.

[Par] Partial Order Trace Analyzer (POTA). http://maple.ece.utexas.edu/˜sen/POTA.html.

[RR98] S. L. Ricker and K. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Transactions on Automatic Control*, 1998. Submitted.

[RW87a] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal of Control and Optimization*, 25(5):1202–1218, 1987.

[RW87b] P.J.G. Ramadge and W.M. Wonham. "Supervisory control of a class of discrete event processes". *SIAM Journal Control Optimization*, 25:206–230, January 1987.

[RW89] P.J.G. Ramadge and W.M. Wonham. "The control of discrete event systems". *Proceedings of IEEE*, 77(1):81–98, January 1989.

[RW92] K. Rudie and W. M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.

[RW95] K. Rudie and J. C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40(7):1313–1319, July 1995.

[SG02] A. Sen and V. K. Garg. Detecting temporal logic predicates in the happened before model. In *International Parallel and Distributed Processing Symposium (IPDPS)*, page to appear. IEEE, 2002.

[SG03] A. Sen and V. K. Garg. Partial Order Temporal Analyzer (POTA) for Distributed Programs. Technical Report TR-PDS-2003-004, PDSL, ECE Dept. Univ. of Texas at Austin, 2003. Available at http://maple.ece.utexas.edu/.

[SL91] R. Sengupta and S. Lafortune. Optimal control of a class of discrete event systems. In *Proceedings of 1991 IFAC symposium on distributed intelligence systems*, pages 25–30, Arlington, VA, August 1991.

[SRA03] K. Sen, G. Rosu, and G. Agha. Runtime Safety Analysis of Multithreaded Programs. TR UIUCDCS-R-2003-2334, Univ. of Illinois at Urbana Champaign, April 2003.

[SS95] S. D. Stoller and F. Schnieder. Faster Possibility Detection by Combining Two Approaches. In *Proceedings of the Workshop on Distributed Algorithms (WDAG)*, volume 972 of *Lecture Notes in Computer Science*, pages 318–332, France, September 1995.

[Tak98] S. Takai. On the languages generated under fully decentralized supervision. *IEEE Transactions on Automatic Control*, 43(9):1253–1256, 1998.

[TC96] K. C. Tai and R. H. Carver. *Testing of distributed programs*, chapter 33, pages 955–978. McGraw-Hill. (A. Zoyama Ed.), 1996.

[TG99] A. Tarafdar and V. K. Garg. Software Fault Tolerance of Concurrent Programs Using Controlled Re-execution. In *Proceedings of the 13th Symposium on Distributed Computing (DISC)*, pages 210–224, Bratislava, Slovak Republic, September 1999.

[TG01] A. Tarafdar and V. K. Garg. Predicate control in distributed systems. *Journal of Parallel and Distributed Computing*, 2001. under review.

[WH91] Y. Willner and M. Heymann. Supervisory control of concurrent discrete-event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[YL00] T. Yoo and S. Lafortune. A generalized framework for decentralized supervisory control of discrete event systems. In *Proceedings of 2000 International Workshop on Discrete Event Systems*, Ghent, Belgium, 2000.

[YL01a] T. Yoo and S. Lafortune. On the computational complexity of some problems arising in partially-observed discrete event systems. In *Proceedings of 2001 American Control Conference*, Arlington, VA, June 2001.

[YL01b] T. Yoo and S. Lafortune. On the computational complexity of some problems arising in partially-observed discrete-event systems. In *Proceedings of 2001 American Control Conference*, Arlington, VA, 2001.