# Project Survey, fundemental distributed algorithms Inria 2017

Stewart Grant

## 1 Specification and verification of Dynamic Properties in Distributed Systems

***Ozalp Babaoglu and Micheal Raynal ISIRA rennes***

### 1.1 Notes

interesting point early on that any number of events may be ”irrelevent with respect to a given property” so their model simply ignores all that are not relevent. In the future consider providing a feild or annotation in a messaging block, (such as govector) which would allow the isolation of messages which pertained to a given property. This would allow for a higher degeree of fidelity. In addition it may be possible to associate messages with a given functionality based on their structure.

**run** a total order of events in a distributed system. This is what really hapend, but is unobservable due to clock skew, and an absent centralized clock.

Exampes of predicates which cannot be satisfied by a simple boolean predicate $A \vee B \wedge C$ where $A$, $B$, $C$ are variables.

- Load ain the network is blaances.
- Resource allocation violates the ät most k-out-of -nc̈oncurrent access requirement
- Message delay alone the route for A to B is less than 5ms
- No more than 100 total users logged in to tmachines A, B, and C
- In the computation of Fig 1 $x >= 2y$.

**sequence spec is an ordered array of simple predicates** $SS ::= SP|SS; SP$

Interval controlled sequences are a key component to realizing temporal predicates esentially the process is as follows. An obervation is a set of states $\Sigma$ on which a property can hold. A mentioned in the prior section $SS ::= SP|SS; SP$ is a sequence of simple properties which hold over subsets of an observation. Temporal properties can be constructed by linking predicates. In the form $\phi; [\bar{\Theta}]; \phi$ where all such phi are observations on which a predicate holds.

$\phi_1; [\theta]; \phi$ is satisfied by an observation if there exists a global state $\Sigma^1$ in which $\phi_1$ holds, a late global state $\Sigma^2$ in which $\phi_2$ holds, and $\Theta$ does not hold in the global states in between. Denoted in the form $\phi_1; [\bar{\Theta}]\phi_2$.

This paper makes the contribution of formualting temporal strings of if *interval-constraied sequest* with the synyax $[\bar{\Theta_1}]\phi_1; \ldots; [\bar{\Theta_m}]\phi_m; [\bar{\Theta_{m+1}}]$

Key takeaway from the formulation. The difference between SS_and ICS is that between states $i_n, i_{n+1}$ there exists a $[\bar{\theta_n}]$ which must not hold on that interval. In that way the expressivness of ICS is greater than that of SS.

Two algorithms are proposed for solving SS **Pos** and **Def**. To compute Def all paths through an observation are taken and must show that the $SS$ formula holds. In constrast $SS$ **Pos** is verified simply by finding a single example in an observation on which the formula holds.

### 1.2 Take Away

This paper presented a novel formulation for describing temporal properties in distributed executions. It is very similar to Dinv in that it uses a latice to formulate an execution. However, where Dinv prunes the search space by examining only strongly consistant cuts, this approach dives in deeper, and actually traverses all possible interleavings in order to satisfy a predicate.

If Dinv were to take this approach it would require that a user specify their invariants ahead of time, if not then all possible observations of a run would need to be fed into Daikon.

Of specific interest is their formulation of $[\bar{\theta}]\phi$ in that it requirs some invariant to not hold before a know property in a given state. Using Dinv’s ground states this same technique could still be applied, although it would be equally restricted by ground states. However, this could still be a potential extension.

The clear problem with mining invariants using this technique is that it is a search, therfore any attempt to infer invariants would result in inferring on the entire search space.

This approach does not take into account failures, it could also be interesting to extend the expressivness of the formula to accept failures and restarts. The hardest part in this would be modifing $\Sigma$ to accept "Global" predicates on a subset of nodes. Further it would be interesting to see if predicates could be defined which allows for the recovery of a node, by a new one introduced to the system.

Overall this is just a scheme for temporal logic detection in a system instrumentd with vector clocks. It could be used for that golden key value spec $\forall get(x) \leftarrow put(x)get(x) == get(x)$ untill $put(x')$. This would have been a cool path to go down if it had not allready been done.

## 1.3 Questions

- Is there a way to divide temporal search paths for mining?
- Is there a way to identifiy identical search paths and merge them for compression?
- Is there an obvious or non obvious way to trim a search path, or limit the lenght of a search for temporal invariant mining?
- Can Search Paths be presented to daikon in a way which preserves temporal logic?

# 2 On the Fly Testing of Regular Patterns in Distributed Computations

**Eddy Fromentin, Michel Raynal Vijay k Garg, Alex Tomlinson**

## 2.1 Notes

This work aim to detect regular patterns in the exectuion of distrubted systems. The first technique to note is that they reduced a distributed execution down to only events corresponding to msg receves and sends. This is the same intuition I had when initially building GoVector, and insturmenting systems for Dinv but it is done formally.

Their technique relies on labling events with a symbol in an execution language. Series of events form automata which are describable by regualr expressions. Not sure where this paper is going, but I assume that they are going to describe a partially orded regex syntax.

This paper did not go much further, the algorithm for performing on the fly regular expressions is relativly simple. Essentally every process maintains an array of the regular expression. The expression nessisarily begins at the beginning of the computation, with omits much of the complexity incurred in the *specification and verification of dynamic properties in distributed computation* paper. Each process maintains a boolean array of the properties in the regex. When a new event occurs locally the process checks if it violates the regex. Otherwise two seperate cases occur. 1) a sent message is received or a local event happens, merge the two and check if they violate any of the regex conditions. On a send event just treat it as a local event. In addition to vector clocks the array of regular experession must be added to messages.

## 2.2 Observations

I think that regular expressions on states are interesting, but ultimatly not going to play into my work. These models could be usefull for specifing something like TCP where the execution is known to be finite, or should be. In typical system cases though I believe that specifications need to be more expressive than regular expersions. For instance Coq and TLA+ the key languages in specification are fully fledged programming languages.

What this work does contribute is an online way to process regex. I think that this is usefull for a couple of reasons 1) I think that it could be key in generalizing some well known protocols such as leader election, where the set of known messages would fit in the vocabulary of events.

I should do a bit of follow up on this work to see if anyone built a tool which automatically translates regex into protocols rather than check a protocol with regex, but I don't think that it would yeild interesting work.

# 3 Shared Global states in Distrubted Computations

**Eddy Fromentin, Michel Raynal**

## 3.1 Notes

The paper begins with the same notation used in prior papers defining partial order and such. The key part of the paper is *shared global states*, which are an element in the lattice of global states such that

Let $\Sigma = (s_1, \ldots, s_i, \ldots, s_n) beaglobalstate$
$\Sigma isshared \iff$

$$\forall(i, j) :: (\text{prev}(s_i) \rightarrow \text{next}(s_j)\ or\ s_j = s_j^{last} or s_i = s_i^0)$$

Essentially a shared global state is a state which has the entire global history as its transitive predisessor, or at least all of the history it could known about transitivly.

The algorithm near the end of the paper shows how shared global state can be detected on the fly, however the algorithm is centralized with diminishes how interesting it is. That stated, shared global state is near conceptually with common knowledge, in that it is the nth degree of eveyone knows everything, it is the case that someone knows everything [1].

### 3.2 Observations

While is is kind of cool that the idea of *Shared Global States* has been formalized here, there is a big want with reguard to purpose. Specifically when would you want to use shared global state. I can only think of Chain Replication. [2]. While shared global state is not a central tenant of the system it would acheive it quite often during execution.

With reguard to Dinv this is almost certanly not the path to go down, SCC are allready too much of a restriction as they do no even consider the world of *observations*. With that said GSS could be an interestin root at which to perform some sort of analysis. It serves as an obvious break point.

If I wanted to infer arbetrary temporal properties on a distributed execution I could for instance make users provide a template, search for observations from GSS roots, enumerate them in a key value store, overlay matching or equivalent states (there would be few, but some may match the template.

This is probably not the way to go either, the size of the output would be gigantic, just note that **GSS can be used for roots of computation** as they are as close to common knowledge as you are ever going to get (and it can be done on the fly)

### 3.3 Questions

- Is shared global state sufficient to coordinate locks? If not what information is required to allow for locking?
- How frequently do these happen in practice, for example in etcd? If the answer is kind of frequently there may be some interesting work to be done.

## 4 On-the-fly analysis of distributed computations

**Eddy Fromentin, Claude Jard, Guy-Vincent Jourdan, Michel Raynal**

### 4.1 Overview

This paper proposed a method for checking properties of distributed computations during exectuon. The Idea is a bit suble, and very general. Given a standard model of a distrubuted systems ie $P_0 \ldots P_n$ processand with events $e_0 \ldots e_m$ apply to each event a work or subset of an alphabet $\Sigma$. As control flow passes through events on different machines check the current alphabet againts a specified automata or regular expression. If an event fails to correlate with the regular expression then it has not met the specification and fails. Otherwize the letters of the event are appended to the message, and upon the next event it is checked againts the regex again.

There is a little bit in the beginning about simplifing a distributed execution down to an LPO. The graph (Figure 1b) does not make a lot of sense. It seems to trim out long messages, and only be concerned with the longest message path through the execution. It was not totally clear to me why this would be used, but I guess the idea is that newer messages have some sort of precident, and can invalidate old messages??

## 5 Efficient Distributed Detection of Conjunctions of Local Predicates

**Michel Hurfin, Masaaki Mizuno, Michel Raynal, Mukesh Sighal**

### 5.1 Notes

The collapastion of internal events, ie the job done by the key-value store in Dinv is refered to here as an interval. More precisely "the $x$'th interval of proces $P_i$ denoted by $\theta_i^x$ is a segment of the computation that begins at event $e_i^x$ and ends at $e_i^{x+1}$. This definiation may be usefull in describing the usage of the key value store, and this citation, along with TODO ►*on the fly analysis & shared global state and others . . . would be usefull. Thanks Raynal*◄

In early Dinv I defined a total ordering on cuts. Although I never ended up using it becuase it was too course grain the same dependency is defined here. Formally: A dependency relation denoted by $\rightsquigarrow$ is defined over the set of all consistent cuts as follows. Let

$C^x = \theta_1^{x_1}, \theta_2^{x_2}, \ldots, \theta_p^{x_p}$ and $C^y = \theta_1^{y_1}, \theta_2^{y_2}, \ldots, \theta_p^{y_p}$ be two consistent cuts, then: $C^x \rightsquigarrow C^y \iff (C^x \neq C^y) \wedge (\forall k, 1 \leq k \leq p, x_k \leq y_k)$

This algorithm is very technical, but somewhat similar to Dinv. Essentailly it aims to show that at some point in relative time a conjunction of local predicates held. The algorithm involves two vector clocks and a boolean vector denoting predicate satisfiability of on a process.

I think the algorithm goes like this. If a predicate is true, set $P_i$ to true. If you get a message with $P_j$ = true update the predicate array. Mark the current vector time, and keep track of the first point in time the predicate was denoted true. If $\theta$ becomes false set all incomming and outgoing predicate messages to false. The predicate becomes true if there is a causal path across all processes such that $\forall i \theta_i = true$. It is not super complicated, the hard part is really just that it is done on the fly.

I think that this algorithm is a bit pessamistic, using a lattice I think that these predicates could be detected more easily.

## 5.2 Observations

This work is very specific. It aims to detect global predicates online, with significant computational overhead. I think that the key takeaway from the algorithm is the technique for detecting causal intervals online. Essentially taint tracking and merging across multiple processes to show that some property $\theta$ is satsified globally.

This work seeks to show that a given predicate is recognized globally and does not have any interaction with a state machine. Could this online technique be used as an atom to verify sequences in in computation, and allow for decentralized distributed computation? For instance would it be possible to encode a bunch of tasks, and have a single process issues a continue message once the task is complete? If the satisfaction of a given task is monotonic then I believe that the answer is yes.

## 5.3 Questions

- Could this technique be used to easily distribute SAT solving? my inital instinct is no, because there is no obvious way to do back propagation.

- Is this applied anywhere? For instance this could be used to show something like termination in Hadoop withought a specified leader. With that stated it would be a bit of overkill.

- could this be used to simplify termination in graph processing? Is this this a more robust solution to predicate dection than the *White/Black* graph coloring that pregal uses.

- Is there a recovery mechanism for satisfied predicates? It seems like in the worst case only a single process will learn that the predicate is satisfied.

# 6 Detecting Atomic Sequences of Predicates in Distributed Computations, 1993

**Hichel Hurfin, Noel Plouzeau, Micheal Raynal**

## 6.1 Notes

The goal of this paper is to detect sequences of atomic properties. I assume that these would be something similar to the detection of distributed mutual exclusion. The key inisght that was presented in the abstract is that these properties on not monotonic. Perhaps this means that atomic properties may hold on the **POS** predicate? I'm not sure because for the sake of guarentted atomicity **DEF** seems to be a nessisary constraint.

Key note, the definition of "Atomicity" in the case of this paper is a sequence of events, of which none violate a specific preperty ie $CA = \neg NP_1 \wedge NP_2 \wedge \ldots \wedge NP_n$. Where each $NP_i$ denots the property on a select processes.

The real purpose of this paper is to count the number of causal paths which satisfy a prefix of the form $do_0; [don't_0] do_1; [don't_1] \ldots do_n$. This differs from the specifing systems paper a little bit, essentially in the way that they are counting the number of occurances.

## 6.2 Observations

An anti observation is, what is the purpose of this algorithm. It seems to be a gneralization of the specification language, defined in Specification and Verification of dynamic properties, but it is simply a counting function. I suppose that the interesting bit comes from handing cases were predicates are invalidated due to being non-atomic however, it was unclear to me why a counter of these properties was wanted and not simply a flag.

This algorithm is extreamly clean. It would be interesting to see if it could be extended to define a generalized decentralized computation framework similar to Dryad but with a decentralized compute. Essentially all computations would be defined as a set of predicates and antipredices, for instance, compute a property on some large graph, and dont compute where any other node has. Instances of conflict could be counted and reset. Individual functions could be specified in this way, and entire programs could be written as a heirarchy of predicates, and loops defined on them. Figuring out how to work in conditionals would be a problem onto itself.

### 6.3 Questions

- What common distributed debugging tasks actually require counts of invariant violations.
- Is it reasonable to define atomic predicates, with the execption of locking?
- Can this be leveraged in reverse, and turned into a control mechanism, rather than a checking algorithm.

## 7 Some Optimal Algorithms for Decomposed Partially Ordered Sets

**Vijay K. Garag**

### 7.1 Notes

I'm reading this one as both an introduction to Garag and as a bit of a break from the proof heavy papers.

This paper allready seems like it is going to be a good resource, it has an algorithm for detecting consistant cuts $n^2m$ time . . . if this is really the case, it may be possible just cut the crap on dinv and really speed it up.

As usual my hopes are dashed this is an early algorithm which only detects a single consistant cut, not all of them like I had wanted. However, the online variant may be useful for making Dinv a streaming service at some point.

The second algorithm in this paper checks if a set of given antichains compose are constrained by a total ordering. The algorithm does so in $O(mnlogn)$ time. Essentially it continuously merges sets of antichains untill it finds aconflits where some set of antichains has events / elements $s$ and $t$ such that $s||t$. It is a very nice algorithm, but given that Dinv never encounters total orderings I don't think that it will come in handy.

### 7.2 Observations

The online detection of consistant cuts could be used as a fron end for dinv easily, but the cost of checking all consistant cuts is still exponential.

### 7.3 Questions

- What is the purpose of finding largest antichains? Potentially it shows that the largest number of chains which can be composed is $n$ but it is still not clear where this would be applied. Is this to show the complexity during distributed debuggin?

## 8 Predicate Detection for Parallel Computation with Locking Constraints

**Yen-Jung Chang, Vijay K. Garag**

### 8.1 Notes

Initially this paper proposes an analysis techinique which analyzes an exponential number of posets in order to check if locking predicates hold. It seems as if it is going to build posets out of code itself.

**After getting through the first 5 pages I was very lost. This paper it seems is too theoretical for me.** TODO ▶*Return to this paper after reading 15 more theoretical papers by Garag*◀

## 9 Modeling, Analyzing and Slicing Periodic Distributed Computations

**Vijay K. Garag, Anurag Agarwal, Vinit Ogale**

### 9.1 Notes

This paper proposes an anaylsis technique for decomposing distributed exectutions. It proposes a very interesting model, in which DAG's are infinine. Their proposed infanite DAG is reffered to as a *d-diagram* and essentially models looping computation post predicate on an infanite timeline. They use this model to verify temporal properties such as liveness.

Not the useage of *cut frontier*. This is proper usage and should be intagrated into the Dinv paper if possible.

A very interesting addition on page 14 is the formalization of recursive vector closk. It is simple algebra, but it states simply that given some initial configuration $C$ lets say, and a continuing recursive computation, the vector clock values for all events, of future computations can be computed a priori.

The paper continues by formulating how to verify liveness properties

### 9.2 Observations

My immidate impression of this paper is that it falls short in its model of a distributed execution. It only defines an infanite execution as a prefix followed by a periodic set of recurrent transitions. It seems that this model is strictly limited to a single periodic computation, and not a general computation which must adapt to interaction.

For instance this model could be used to analyze a key value store which only serverd puts, or gets, or a fininte

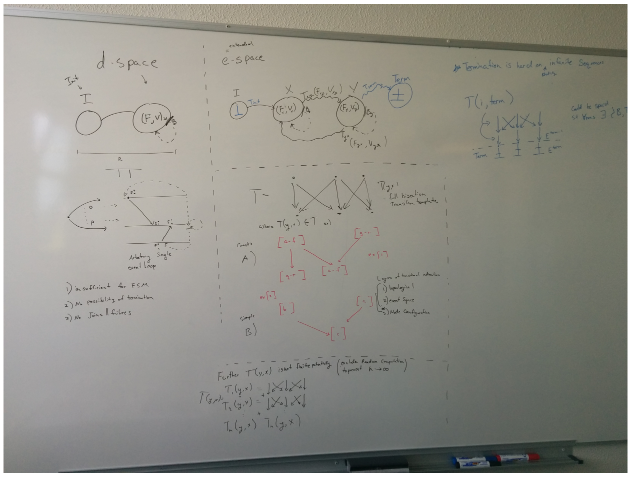
d-space
Init
I
(F,V)
B
R
Arbitrary Single event Loop
1) insufficient for FSM
2) No possibility of termination
3) No Joins ‖ fibres
extended
e-space
I
Init
X
Y
Term
$(F_x, V_x)$
$(F_y, V_y)$
$T_{yx}(F_{yx}, V_{yx})$
T=
T(y,x)
- full bisection Transition template
where $T(y,x) \in T$ ex)
Complex
A)
[a-f]
[g-r]
[a-f]
[g-r]
ex[1]
ex[*]
[b]
[c]
[c]
Simple
B)
Layers of transitional indirection
1) topological
2) event Space
3) Node Configuration
Further T(y,x) is not finite potentially (exclude Random Computation to prevent $h \to \infty$)
$T(y,x) = T_1(y,x) + T_2(y,x) + \dots + T_n(y,x)$
Termination is hard on infinite Sequences
T(i, term)
Term
Could be special st ∀nms ∃ {B,T

**Figure 1:** Sketch of a potential extension to the d-diagram model

repeatable sequence of accesses. It would however fall short in the following scenarios.

I was interested in extending the model presented in this paper. As mentioned in the prior paragraph there are a few shortcommings, such as the models ability to only quantify a single infanite state. I drafted an extension to this model on the board. TODO ►*Take a photo of the whiteboard and insert here*◄

- addition of new nodes
- failure of a node
- loss of messages *I think?*
- non deterministic accesses
- Recovery / Fault detection / Reconfiguration

I believe that a more complete model would generalize *d-diagrams* to be similar to lego blocks, or components of a generalized state machine of a computation (but I'll finish the paper before jumping on this).

# References

[1] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *J. ACM*, 37(3):549–587, July 1990.

[2] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 7–7, Berkeley, CA, USA, 2004. USENIX Association.