

Dynamic Consistency Analysis for Convergent Operators

Alva L. Couch and Marc Chiarini

Tufts University, Medford, Massachusetts, USA
alva.couch@cs.tufts.edu, marc.chiarini@tufts.edu

Abstract. It has been shown that sets of convergent operators with a shared fixed point can simulate autonomic control mechanisms, but many questions remain about this management technique. We discuss how an autonomous agent can reason about whether its convergent operators share a fixed point with the operators of other agents. Using a concept of time based upon operator repetition, we show that a failure to achieve convergence within specific time limits can be used as a probabilistic indicator of inconsistencies in local policy. We describe a statistical inference technique that determines if an agent’s promise strategy is feasible. The strengths of this technique are that it is both scale-invariant and exterior to the operators whose consistency is being evaluated.

1 Introduction

How does a configuration agent in a highly complex and distributed network reason about the workability of its choices? What reasoning techniques work best in a situation in which agents have only local information? What strategies other than global information exchange can help agents reason? We attack these questions in a novel way.

This paper arose from a question asked at AIMS 2007 by Jan Bergstra: “How can you be sure that your [operators are] consistent?”. By this, he meant “logical consistency”, i.e., that the goals of the operators cannot embody a contradiction. We thought about this question for some time, and then decided upon a novel response: *logical* consistency of operators is useless in a ubiquitous computing environment, and *statistical* notions of consistency make more sense and are more useful.

To start to understand what consistency might mean in a ubiquitous computing environment, we combine and expand two threads of prior work: convergent operators and self-organizing precedences.

An *operator* is a management operation on a host or network. A *convergent operator* P is an operator with the specific behavior that – when repeatedly applied to a host or network – modifies the host or network so that it exhibits a state in some known set S_P of desirable states. A convergent operator P is *idempotent* on states in its result set S_P , i.e., it does not change any state that

is already considered desirable. Thus every $s \in S_P$ is a fixed point of P , i.e. $P(s) = s$, and every convergent operator is also a *fixed-point operator*.

Convergent operators can take many forms, e.g., one operator might prune disk space for users who are over quota, while another might act on process space to prune runaway processes, and a third might modify the number of threads in a web server to optimize response time.

Prior work shows that autonomic computing can be “approximated” by a collection of convergent operators applied repeatedly at random, provided that these operators all share some common fixed-point state [1]. Each operator – in effect – embodies its own isolated control loop, and the set of operators behaves like a composition of multiple control loops.

2 Operator consistency

To understand what consistency might mean for a set of operators \mathcal{O} , we point out that the set of fixed points S_P of an operator P is a representation of the *policy* of P . While we might be accustomed to expressing policies as a set of logical rules R_P , the set of states S_P that happen to obey those rules is a reasonable (though more sizable) substitute. Two sets of rules R_P and R_Q are consistent if their structure does not contain a logical contradiction. By analogy, we define:

Definition 1. *Two operators P and Q are consistent if the intersection of their fixed point sets is nonempty.*

Note that trivially, operator consistency is equivalent with rule consistency, in the sense that states matching contradictory rules have no intersection, and vice-versa ¹. Likewise,

Definition 2. *A set of operators \mathcal{O} is consistent if the intersection of all fixed point sets of operators in \mathcal{O} is non-empty.*

One must also ensure that consistency is practical as well as theoretically possible:

Definition 3. *A set of operators \mathcal{O} is reachably consistent (with respect to a set of baseline states \mathcal{S}) if, through random applications of operators in \mathcal{O} , starting at a state in \mathcal{S} , the operators always achieve a common fixed point.*

If all operators are at a fixed point, then clearly their policies do not contradict one another, so reachably consistency is a sufficient condition for logical consistency. The converse, however, is not true. There may be a consistent state that, though it be fixed for all operators, can never be achieved. Suppose, e.g., that one operator’s policy is that the web server serves subdirectories of students’ home directories, but that no operator exists to actually set up a web server. Then the desired state is consistent with that of other operators, but not reachably consistent, because no operator exists that can achieve that state.

¹ Expanded definitions of consistency are explored in [5].

This notion of consistency is reasonable for a small, known set of operators, but in a ubiquitous computing environment, with no true centralized control, there is no clear notion of which operators are active at a given time. The contents of \mathcal{O} is a moving target. From the point of view of any specific agent, the total set of operators and policies in effect *cannot* be known, because any snapshot of that state could potentially have changed since the snapshot.

Thus the concept of logical consistency – while well-defined – is simply not very useful. A new notion of consistency is needed; one that is useful to an agent with incomplete knowledge of the space of operators and/or policies being enforced.

3 Operator Precedence

Convergent operators only work properly if the precedences between operators are satisfied. For example, one operator might mount a filesystem, while another might start a service depending upon that filesystem. We refer to an active operator whose preconditions are fulfilled as being *operative*; an active operator with at least one unmet precondition is *inoperative*.

The effects of operators often depend upon precedences between them. For example, consider operator O_1 that mounts a filesystem on a host, and operator O_2 that sets up a web service on that filesystem. These operators have a precedence relationship: O_1 must precede O_2 . Once O_1 completes its task, it has no further effect unless some *other* entity unmounts the filesystem, in which case O_1 mounts it *again*. Thus the mounted filesystem is a *fixed point* of operator O_1 . Once this fixed point is achieved, operator O_2 can achieve its fixed point and we conclude that the set $\{O_1, O_2\}$ has a fixed point as well. The result of both operators is an *emergent* fixed point that is fixed for both operators, but fixed in *different ways* for each operator. Thus each operator can be thought of as acting on an *aspect* of the network, and the composition of operators into a set can be viewed as similar to *aspect composition* as defined in [6–8].

By contrast, consider an operator O_3 that removes any web service present. The set $\{O_1, O_2, O_3\}$ has no fixed point, because O_2 and O_3 *conflict*, and are thus *inconsistent*. This is analogous to a set of policy rules that are logically inconsistent. Anything O_2 does, O_3 undoes, and vice versa. Note that conflicts can be more subtle; it is just as damaging if O_3 simply breaks O_2 's web server so that O_2 is forced to continually repair it. As another example, consider, e.g. operator O_4 that moves the service set up by O_2 to another filesystem. This can only happen after O_2 achieves a fixed point. Does O_4 conflict with O_2 ? This depends very much upon how O_2 is defined, and the difference between what it *assures* and the states it *accepts* as conforming to its needs. It is possible that a set of operators is consistent even though no consistent state is *reachable* via application of the operators.

As a first step in understanding precedences between operators, [2] shows that if operators are applied serially, and each operator is aware of its own needs, then it is possible to satisfy the precedences between operators *without* codifying the

precedences separately or centrally. Instead, one applies n (distinct) operators in sequence n times. Since this sequence contains every permutation of the operators, it contains at least one permutation satisfying the actual precedences of the operators. If the operators are constructed to do no harm unless their preconditions are met, then all operators will become operative in the course of $O(n^2)$ trials, where n is the number of operators. We will call this result the *maelstrom theorem*, after the cyclic “whirlwind” motion of operators in the proof.

Reasons for this perhaps counter-intuitive result are twofold: facts about permutations and assumptions about operators. Operators are assumed to be *aware of their precedences* and *idempotent* unless precedences are fulfilled. An operator, applied to a network that is not ready for it because preconditions are lacking, will not affect network state or other operators. Likewise, an operator applied to a network to which it has already been successfully applied will do no harm. Changes will occur only in the case where an operator’s preconditions have been fulfilled and the network does not already conform to the operator’s expected outcomes.

This is the context in which prior results end and this paper begins.

4 Consistency as an Emergent Property

So far, we have translated the problem of rule consistency into the problem of determining whether fixed-point sets intersect. This change does not yet simplify matters. It remains difficult to analyze whether a given set of operators shares a fixed point, particularly when the operators act upon different parts of a distributed network. Static analysis of operator fixed-points is difficult enough when we have complete knowledge of the operators (the problem is equivalent to policy consistency[3]), and virtually impossible when we have incomplete knowledge of the operators[4]. Based upon prior work, *without further assumptions*, we conclude that the problem of statically determining whether a particular set of distributed operators share a fixed point is *intractable*, in the same way that it is computationally hard to determine consistency of an unconstrained set of rules.

But the maelstrom theorem described above provides a possible alternative to this quandary. If we know that a hidden order (e.g., a total ordering, or logical consistency) *must* emerge in a certain number of steps of a process (or in a known time interval), and it does *not* emerge, then *it must not be present*. If we can then determine *when* that order should emerge, then its emergence is a necessary and sufficient proof of its *existence*, while lack of emergence within time limits constitutes statistical (but not deterministic) proof of its *non-existence*. We call this last statement the *emergent ordering principle*.

The emergent ordering principle serves as a starting point for a very different notion of consistency than before. Prior attempts at determining consistency (of operators or – equivalently – of the policies that they enforce) relied upon describing the *intent* of operators (or, equivalently, the constraints of policy). The emergent ordering principle conveniently circumvents this requirement; we

need not compute a total order of operators in order to know that such an order exists. Similarly, a proof of the existence of a set of consistent operators need not inform us as to exactly how or why they are consistent.

Perhaps most important, whether consistency and order emerge globally is relatively unimportant; what is important is whether consistency and order arise from one's own actions or not. From an agent's point of view, there is no meaning to global consistency; what the agent manages is either consistent or not. Thus we study how to test for consistency *without* describing intent, computing global information, or deriving causal information.

5 Kinds of Operators

In describing the properties of statistical consistency, there are many kinds of convergent operators and it will help us to consider each kind of operator separately.

Every result in this paper presumes the existence of a set of convergent operators, each of which has one or more fixed points, in the sense that once some fixed point is achieved (through some finite number of applications of the operator), the operator makes no further changes to the network unless some other force moves the network away from the operator's fixed point.

Many common operators are *single-step*, in the sense that they perform only one change if their precedences are fulfilled, and are idempotent on both sides of that step. Examples of a single-action operator include "mounting a filesystem" or "setting up a service". A single-step operator acts upon a non-conforming part of a network, checks that necessary preconditions have been fulfilled, and then effects a state change in that part to make it acceptable.

A "multiple-step" operator, by contrast, takes several steps to achieve a fixed point. Examples of a multiple-step operator include incrementing (or decrementing) the number of threads for a web server until response time is optimal. In this case, each operator invocation adds one thread, and the fixed point is achieved when a desired number of threads w (according to some externally defined criteria) have been invoked. This is one example of a *bounded multiple-step operator*, in which the number of invocations before achieving a fixed point is bounded by some constant L . In this case, L is the absolute bound on the number of threads the web server can support, which is always finite. Common "autonomic" operators, e.g., performance tuning, are all multi-step operators.

Note that neither of these definitions truly conforms to the definition of Burgess, who allows an operator to function in a continuous domain[1]. Burgess' definition of a fixed point operator is one whose result approaches a fixed point as the number of operator applications increases, possibly *without* bound. By contrast, we require an operator to approach a fixed point after a *finite and constant* number of iterations L . Thus, the results below cannot be extended to operators under Burgess' definition.

6 Fixed Points and Policies

Note in the above examples that the existence of a fixed point for a set of operators is in some sense *semantic rather than syntactic*. Whether O_2 and O_4 conflict depends upon what they *mean* or *intend*, rather than how they are *coded*. While the reader may suggest that they could be coded such that the fixed point of a set becomes a syntactic property, this would not simplify the problem of determining whether a fixed point exists. The reason for this is that the codings – and their semantics – are *distributed* in a network, and centrally collecting that information in a ubiquitous network is both intractable and unreasonable.

There is a strong relationship between “operator fixed points” in the semantic view of network management, and “policies” in the syntactic view. Presumably, the reason for constructing an operator is to enforce some (perhaps vaguely defined) policy. A policy, by contrast, is a very specific description of what should happen, independent of how it might be arranged to happen.

While it is quite obvious that one can construct an operator to implement a policy, the converse is not so obvious. By nature, an operator is imperative, while a policy is declarative. The difficulty arises in codifying what an operator does in declarative terms. For example, consider an operator that – through some complex and unknown process, perhaps enabled by machine learning – decides which user processes to kill. This is a perfectly usable operator, but might not be possible to codify in a policy by any more useful language than “do what this operator says to do.” Thus we consider operators to be a more general mechanism than policies for managing network function.

We diverge from current work on policy consistency and take a new path. The problem of policy consistency is to determine – given a syntactic description of the desired fixed points of a policy – whether those fixed points conflict or not. Instead, we step *sideways* (figuratively speaking) and avoid syntactic coding of intent entirely. For us, consistency of a policy expressed as a set of operators means that they have a *semantic* fixed point that *emerges* over time, and is represented by a network’s *state and behavior*. Operators are considered consistent if they together *achieve* a fixed point, regardless of whether we understand *how* they achieve it, and independent of whether we can even *codify* their intent.

The reason for this seemingly bizarre world-view comes from wanting to be able to analyze very large networks. In a ubiquitous network consisting of billions of nodes, determining “how” is intractable, while techniques for determining consistency may remain tractable.

7 The Quandary of Observability

When dealing with configuration management – and operators in particular – lack of observability is a common issue[9]. In the above, we are not assuming there is any concept of a globally observable action; in particular there is no way to *observe* global consistency. Agents reason *on their own* and *in isolation*. It is quite possible that each agent manages only one operator. The definition of an

inconsistent set of operators is thus *relative to the observer*. Suppose, e.g., that each operator is applied by a different agent. It is plausible that one agent would discover an inconsistency that another could not observe, because the agent only sees that *its own operator* does not approach a fixed point.

As an example of this, consider a simple case in which one operator O_1 mounts an external filesystem and another operator O_2 establishes a web server on it. Suppose that the agents applying these operators cannot otherwise communicate. Suppose further that O_1 never achieves a fixed point, perhaps because the filesystem it desires to mount is improperly exported. Then from the point of view of O_2 , the set of operators is inconsistent, while from the point of view of O_1 , it is still consistent, because O_1 has never been able to observe any inconsistency. The fact that its poor view of the world results from its own lack of capability is not of consequence in the definition of consistency.

We handle this quandary as follows. The agents for which consistency is observed (by approaching a common fixed point) are the “haves”, while the agents that never achieve a fixed point are “have nots”. An agent who – from its own point of view – has achieved its aims, has no problems. It is the “have nots” that need to work on different strategies to achieve their aims. In any sufficiently complex system, there will always be *some* “have nots”; the goal is to minimize these and not allow them to dominate the network.

8 Synchronous Operator Activation

Our first exploration is to understand what happens in an environment in which operators can be applied in an ordered (synchronous) fashion. Results for synchronized clocks mimic the results in [2], except that non-emergence of order is now a distinct possibility.

Theorem 1. *Suppose that n single-step operators O_1, \dots, O_n are to be applied in sequential order. Suppose that the longest precedence chain in the n operators consists of k operators. Suppose that one repeats the sequence k times for a total of nk invocations. If there are no external effects, and a common fixed point has not been reached for all operators, then the operators are inconsistent.*

Proof. This is in essence the same result as stated in [2], except that this theorem is stated in converse form and adds the constraint that there are no more than k operators in any one chain of precedences.

The key to the theorem is that nk invocations contain all $k!$ permutations of the k operators in each precedence chain. Thus after nk invocations, all permutations have been tried, and if a fixed point has not been reached, then there is no permutation that produces consistency, and we can conclude that the operators are not consistent.

To demonstrate this claim, note that in each of k blocks comprising n steps, all n operators are tried. Thus one can construct an arbitrary permutation by choosing one operator from each block of n invocations. Thus nk iterations are sufficient to try all permutations. \square

It is important in the above proof that there are no functioning external operators about which one has no knowledge. If one is testing the consistency of $\{O_1, O_2, O_3\}$ and – unbeknown to us – an operator O_4 inconsistent with this set is sporadically being applied, then one may conclude erroneously that the set is inconsistent, even when it is consistent. However, the theorem’s conclusion remains accurate if an unknown operator that is *consistent* with the set is being applied sporadically. This does not interfere with the theorem’s inference in the same way as an inconsistent operator would.

We sidestep this issue in the following sections by changing the way operators are applied. If all operators are applied in the same fashion, and there are unknown operators being applied, then inconsistency is a property of the whole set of operators, not just those that are known. It is not necessary to know how many operators there are and what they are doing in order to know whether they are consistent or not. This is a fundamental strength of this form of analysis.

9 Heartbeats and Time

To apply the emergent ordering principle in the distributed case, we must define what time means in a distributed system. Our definition differs from that of Lamport[10] and researchers involved in distributed performance analysis; we have no need to synchronize the actions of agents.

To define our notion of time, we assume that there is a “heartbeat rate” λ at which, on average, all operators are applied to the network repeatedly. We assume that the time between applications of the same operator follows an exponential distribution with mean $1/\lambda$ and standard deviation $1/\lambda$. As a result, operator application is *memoryless*, in the sense that future frequency of operator invocations does not depend in any sense on past behavior. For now, we assume that the process of applying each operator is asynchronous from applying any other and that all operators share the same application rate λ .

The astute reader will realize that this model matches CFengine’s[11–13] model of operator application; the operators are applied periodically to “immunize” the system against adverse effects[14, 15]. As in Cfengine, it is not possible to assure that an operator occurs at a specific time, but one can assure the *statistical* behavior of an operator as it is applied over time. Our statistical model is chosen arbitrarily and is neither important nor essential, but it is convenient for the arguments that follow. Any other model of operator invocation can be substituted without changing the character of the following results.

10 Fixed Points and Time

The purpose of the “heartbeat” is to provide a coordination point for agents *without* requiring them to communicate. If an agent does not apply an operator at the “heartbeat” time, it is presumed to be down or out of compliance. It is not necessary for the heartbeat to be seen by an agent; one can observe its effect. The reason for this is the following:

Theorem 2. *Suppose we have a set of n single-action operators, repeatedly applied to a network at the same rate λ , where inter-arrival times are exponentially distributed. Suppose that in these operators there are chains of precedence of at most length k . Then the probability that the operators are consistent after time t , given that the network has not achieved a fixed point, is less than $1 - (1 - e^{-\lambda t/k})^{nk}$, which approaches 0 as $t \rightarrow \infty$.*

Proof. The proof utilizes Bayesian relationships between several hypotheses. Let t represent elapsed time since the reference time t_0 at which operators began to be applied. Let H be the hypothesis that the operators are consistent, let $R(t)$ be the hypothesis that all operators have been repeated in all permutations at time t , and let $F(t)$ be the hypothesis that a fixed point for all operators has been observed at time t .

At the outset, no operators have been applied, and we have no information about the likelihood of H except for its relationships with the other hypotheses $R(t)$ and $F(t)$, for which we also initially have no information.² Clearly, $F(t)$ implies H and – since $F(t)$ can only be observed over time – the probability $\text{Prob}(H)$ that the operators are consistent is a time-varying quantity.

If at a particular time t , $R(t)$ is true, and $F(t)$ is false, then H is false by Theorem 1. Thus $R(t) \wedge \neg F(t) \rightarrow \neg H$.

Now suppose at a particular time t that $F(t)$ is false. It follows that the only way that H can be true is if $R(t)$ is false; otherwise H is false by the above. Thus $\text{Prob}(H|\neg F(t)) \leq \text{Prob}(\neg R(t)) = 1 - \text{Prob}(R(t))$. The reason for the inequality is that $R(t)$ is sufficient but not necessary to disprove H . One might be able to infer that H was false long before $R(t)$ is true, e.g., by observing two operators conflicting over a short time.

$\text{Prob}(R(t))$ is difficult to compute, but we can bound it by approximation. Consider the hypothesis $S(t)$ which represents that all n operators have been applied each t/k seconds, so that each operator has been applied k times in t seconds. By Theorem 1, $S(t) \rightarrow R(t)$, so $\text{Prob}(S(t)) \leq \text{Prob}(R(t))$ and $1 - \text{Prob}(S(t)) \geq 1 - \text{Prob}(R(t))$.

To understand this relationship, consider the Venn diagram in Fig. 1. The circles in the diagram represent overlaps in probability space. If one hypothesis implies another, this is a containment relationship. In our case, we know that $S(t) \wedge \neg F(t) \rightarrow R(t) \wedge \neg F(t) \rightarrow \neg H$, which is depicted as a containment relationship between their probability masses.

The probability of S being true, unlike that of R , is easily computed. The probability of $S(t)$ with respect to elapsed time t is the result of k independent sets of events, each of which are composed of n independent events, where each event has probability $(1 - e^{-\lambda t/k})$. Thus the composite probability of these nk events is $\text{Prob}(S(t)) = (1 - e^{-\lambda t/k})^{nk}$, and we conclude that at time t ,

$$\text{Prob}(H|\neg F(t)) \leq 1 - \text{Prob}(R(t)) \leq 1 - \text{Prob}(S(t)) = 1 - (1 - e^{-\lambda t/k})^{nk}$$

² One reviewer felt that the estimates would be greatly improved by use of a priori estimates of the likelihood of H . We agree, but at the outset of this process, we have no such information.

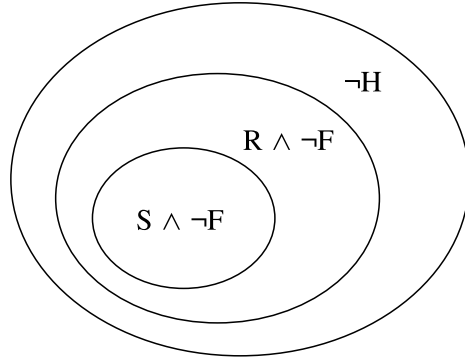


Fig. 1. If all permutations of operators have been tried and a fixed point was not observed ($R \wedge \neg F$), then the set of operators is inconsistent ($\neg H$), but this is not the only way to prove $\neg H$.

The exponential dominates the constant power, so this quantity approaches 0 as $t \rightarrow \infty$. \square

In the theorem, n is the number of operators and k is the maximum dependency between them. In regular practice, the number of operators managing a network might be in the thousands, while the dependencies between them remain rather limited; an average service depends on at most 10 others.³

The practical uses of this theorem are easier to describe than the theorem. By assumption, all operators try to operate on average once each $1/\lambda$ seconds. There is a precedence chain of length k , where n is the number of operators. In this case,

1. After time $1/\lambda$, on average, one has operated.
2. After time $2/\lambda$, on average, a second has operated (depending upon the first).
3. ...
4. After time k/λ , on average, the last in the chain has operated.

On average, within time $k\lambda$, a set of operators is either proved consistent or there is enough evidence of inconsistency to safely assume that the set is inconsistent.

The key to the usefulness of this theorem in practice is that the agent discovering the inconsistency is exactly the agent whose operators should change to react to it. If an agent has been listening for time t , and can be 99% sure that the current set of operators is inconsistent, then it can act to preserve consistency. If we want $\text{Prob}(\neg H) \geq .99$, we know that is true if $(1 - e^{-\lambda t/k})^{n^k} \geq .99$. In turn, this is true when $t \geq -k \cdot \ln(1 - \sqrt[n^k]{.99})/\lambda$. In general, .99 can be replaced with

³ This estimate is based on the combined system and network administration experience of the authors, and is roughly the maximum depth of dependencies exhibited in software packages for Linux distributions.

any confidence limit. A graph of time versus confidence for $\lambda = 1$, $k = 10$, and $n = 10, 20, 30$ is shown in Fig. 2. An alternate view is shown in Fig. 3.

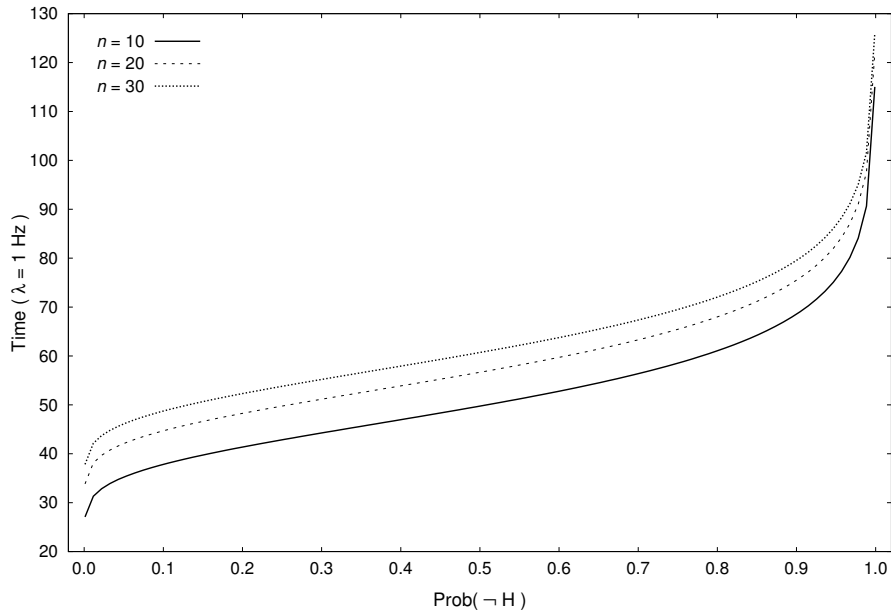


Fig. 2. The probability of inconsistency over time, given that a fixed point has not been observed, for sets of $n = 10$, $n = 20$, $n = 30$ operators containing no precedence chain over $k = 10$ operators long.

Note also that the construction of the theorem is invariant of the particular distribution of operator invocations:

Corollary 1. *If invocations of operators in Theorem 2 are distributed according to a cumulative distribution function $c(t)$ (rather than the exponential distribution), and invocations are independent events, then the probability that the operators are not consistent at time t , given that consistency has not been observed, is less than $1 - c(t/k)^{nk}$.*

Proof. Substitute $c(t)$ for the exponential cumulative distribution function $1 - e^{-\lambda t}$ in the proof of Theorem 2. \square

Note that in our model, a security violation is modeled as an operator; it is just like any other operator, except that we do not have knowledge of its contents. It is typical for such an operator to introduce inconsistencies in policies, by trying to assert states that other operators attempt to prevent. Since our model does not require codification of the intent of each operator, it can be used to model effects of security violations without knowing the exact effect of a violation.

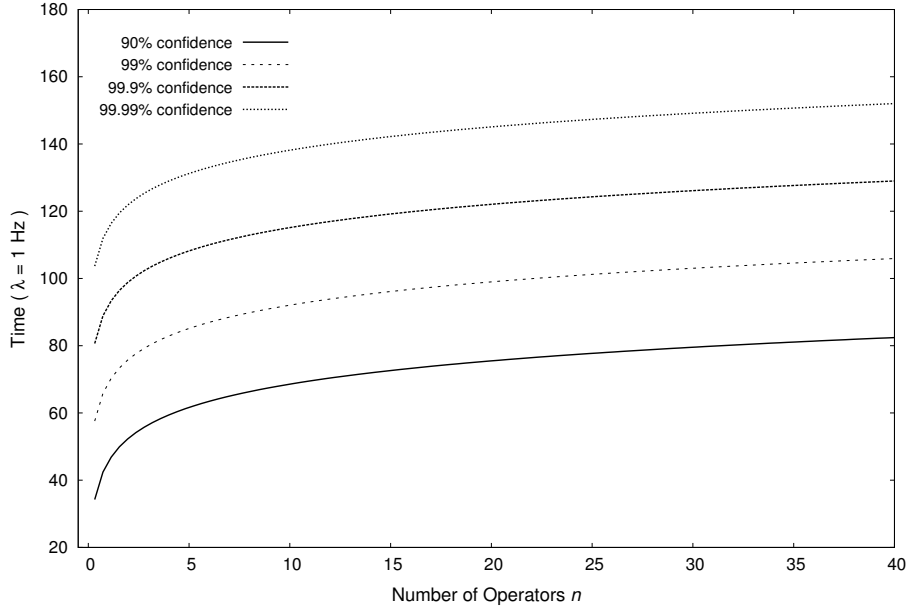


Fig. 3. Time taken to disprove H, given that a fixed point is not observed, for a precedence chain limit of $k = 10$ and increasing numbers of operators.

Thus it is possible to model a security violation as the emergence of disorder in a previously ordered environment.

11 Multiple Time Bases

The assumptions in Theorem 2 are still too limiting for most practical cases. For example, we assume in Theorem 2 that all operators are applied at the same rate. The case in which different operators are applied at different rates is easy to handle:

Corollary 2. *Suppose operators O_1, \dots, O_n are applied at rates $\lambda_1, \dots, \lambda_n$ and that $\lambda = \min(\lambda_i)$. Then Theorem 2 applies to this system with rate λ and, given that a fixed point has not been observed after time t , the probability that the set of operators is consistent is less than $1 - (1 - e^{-\lambda t/k})^{nk}$, which approaches 0 as $t \rightarrow \infty$.*

Proof. Exponential inter-arrival rates are additive; e.g., a process that is the sum of two rates λ_a and λ_b has a composite rate that is the sum $\lambda_a + \lambda_b$. Thus every operator O_i is applied at a rate λ plus (additionally) a rate $\lambda_i - \lambda > 0$. The extra rate does not improve the results of Theorem 2, but it does not hurt, either. Repeat the proof of Theorem 2 to obtain the result. \square

Thus a “common time base” assumption can be utilized without harm for the slowest rate in a set of rates.

12 Bounded Operators

Suppose we have a set of operators that are not single-step, but instead are guaranteed to find a fixed point in at most L steps, where L is a constant. We can still predict the time at which they will become stable, if any, as follows:

Corollary 3. *Suppose operators O_1, \dots, O_n achieve fixed points in at most L steps each. Then we can model this system as containing nL operators, and the probability that the operators are consistent, given that consistency has not been observed after time t , is $1 - (1 - e^{-\lambda t/k})^{nkL}$, which approaches 0 as $t \rightarrow \infty$.*

Proof. Model each operator that achieves a fixed point in L steps as L separate operators, one per step. Apply Theorem 2 to the resulting set of nL operators to obtain the result. \square

Corollary 4. *Suppose every operator O_i has a different constant limit L_i . Let L be the maximum of the K_i . Then the result of Corollary 3 holds for L .*

Proof. Model each operator in the base set as L operators in a new operator set, where some of these are identity operators. The proof above then applies without change. \square

Note that these bounds are loose and can be easily tightened. The important point of these corollaries is that these bounds – and even tighter bounds – are easy to compute.

13 Conclusions

In this paper, we turn the problem of analyzing policy consistency somewhat upside-down, viewing consistency as an emergent property of a self-organizing system of operators that try to implement policy. We demonstrate that there are Bayesian hypothesis-testing techniques that aid one in determining whether consistency is present, and discuss their limitations. But this is just the tip of a much larger iceberg.

First, the Bayesian inference techniques we use are just a small part of the true Bayesian lattice of hypotheses for the problem. We are only considering one way of analyzing the hypotheses, based upon one path through the lattice, and are only concentrating upon upper bounds for consistency hypotheses. The theorems in this paper can be thought of as “existence proofs of upper bounds on the probability of hypotheses”. These are extremely conservative bounds and ignore much information that might be available. There are many other statistical relationships yet to be explored, and the bounds described herein are not optimal.

Second, it is relatively straightforward (though laborious) to extend these results to test whether a new operator is consistent with an existing fabric of operators. This gives the individual agent the ability to make reasoned choices as to whether it should change its management strategy (by changing the operators that it applies). This requires precisely defining the hypotheses to be tested from an individual agent's point of view, and is left for future work.

One impact of this work is that autonomous agents can use bounds in their reasoning processes that have a reasonable mathematical meaning. This paper shows how – in deciding upon a course of action – an agent can determine when it has waited “long enough” for external events. This is a small first step toward reasoning processes with strong statistical properties.

14 Acknowledgments

This paper draws its inspiration from the AIMS community, including Mark Burgess, Jan Bergstra, and many others.

References

1. Burgess, M., Couch, A.: Autonomic computing approximated by fixed-point promises. In: Proceedings of the First IEEE International Workshop on Modeling Autonomic Communication Environments (MACE), Multicon Verlag (2006) 197–222
2. Couch, A.L., Daniels, N.: The maelstrom: Network service debugging via “ineffective procedures”. In: LISA, USENIX (2001) 63–78
3. Lupu, E., Sloman, M.: Conflicts in policy-based distributed systems management. *IEEE Trans. Software Eng.* **25**(6) (1999) 852–869
4. Dunlop, N., Indulska, J., Raymond, K.: Dynamic conflict detection in policy-based management systems. In: EDOC, IEEE Computer Society (2002) 15–26
5. Couch, A., Chiarini, M.: A theory of closure operators. In: AIMS. (2008) (submitted)
6. Burgess, M., Couch, A.L.: Modeling next generation configuration management tools. In: LISA, USENIX (2006) 131–147
7. Anderson, P.: Configuration Management. SAGE Short Topics in System Administration. USENIX (2007)
8. Couch, A.: Configuration management. In Bergstra, J., Burgess, M., eds.: Handbook of Network and System Administration. Elsevier, Inc. (2007) 75–133
9. Couch, A.L., Sun, Y.: On observed reproducibility in network configuration management. *Sci. Comput. Program.* **53**(2) (2004) 215–253
10. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* **21**(7) (1978) 558–565
11. Burgess, M.: A site configuration engine. *Computing Systems* **8**(2) (1995) 309–337
12. Burgess, M., Ralston, R.: Distributed resource administration using cfengine. *Softw., Pract. Exper.* **27**(9) (1997) 1083–1101
13. Burgess, M.: Theoretical system administration. In: LISA, USENIX (2000) 1–13
14. Burgess, M.: Computer immunology. In: LISA, USENIX (1998) 283–298
15. Burgess, M.: Cfengine as a component of computer immune-systems. Proceedings of the Norwegian Conference on Informatics (1998)