# On the Algebraic Structure of Convergence

Alva Couch[1] and Yizhan Sun[1]

Tufts University, Computer Science, Medford MA 02155, USA,
couch@cs.tufts.edu, ysun@cs.tufts.edu,
WWW home page: http://www.cs.tufts.edu

**Abstract.** Current self-healing systems are built from "convergent" actions that only make repairs when necessary. Using an algebraic model of system administration, we challenge the traditional notion of "convergence" and propose a stronger definition with improved algebraic properties. Under the new definition, the structure of traditional configuration management systems is a natural emergent property of the algebraic model. We discuss the impact of the new definition, as well as the changes required in current convergent tools in order to conform to the new definition.

## 1 Introduction

There are at present three main approaches to automated configuration management. The "imperative" approach of ISconf and its relatives[16, 25, 26] is closest to human practice; it expresses configuration actions as a series of commands applied to a system that possesses a known initial state. The "convergent" approach of Cfengine and its relatives[3–6] expresses configuration as a set of known states to assure in an otherwise unknown system. The "generative" approach of LCFG and its relatives[1, 2, 14, 15, 18, 19, 24] controls state via regeneration of individual configurations from an overarching ruleset, rather than by making incremental changes to existing configurations.

The three configuration management strategies differ in their ability to handle reordering of configuration tasks in time. Imperative commands can branch based upon existing conditions created by other configuration management commands; the order of operations always matters in any imperative scheme[16, 26]. Meanwhile, Cfengine's "convergence" proponents argue that not only is order irrelevant, but novel self-healing configuration management strategies can be based upon *random* ordering of convergent actions[12, 17]. Which is the proper strategy: deterministic ordering, random convergent ordering, or generative reconstruction? We think that part of the key to this question lies in *algebraic* properties of the primitive operations for the three kinds of methods.

One problem with current "convergent" strategies is that a composition of individual convergent actions need not be convergent as a composite action. We propose a very simple algebraic model that justifies a new definition of convergence, in which a composition of convergent actions remains convergent. We argue that this definition is "more correct" because the definition allows us to derive an appropriate algebraic structure for an effective self-healing system.

## 2 Semigroups of Actions

We first develop an algebraic model of configuration management. This model adopts some curious conventions in order to make a minimal set of structural assumptions about configuration management. We then allow algebraic structure to arise from the axioms we choose for the system. In this way, we can analyze the axioms based upon the system properties that they imply when used.

Configuration actions are commands or subroutines that change the state of the managed system. We make no other assumptions about their nature. Let $P$ represent a set of actions. Each action $p \in P$ acts upon a system $X$ to produce a change in configuration. Configuration actions might include things like:

- replace `/etc/inetd.conf` with the one at `foo:/bar/repo/inetd.conf`
- replace `/etc/inetd.conf` with the one at `cat:/dog/repo/inetd.conf`
- delete all udp protocol lines in `/etc/services`.
- cd into `/usr/src/ssh` and run `make`.

etc. One can compose actions $p, q \in P$ in the natural way, by performing one after the other from right to left, e.g., $p \circ q \circ X = p(q(X))$. We also use composition notation for the final step $q(X) = q \circ X$ because a system itself is already composed of actions, even those such as "buy system from vendor Y". The system itself is some kind of action (e.g., a recipe for building it) but that action may not be in the set of actions $P$ that potentially *configure* $X$.

To simplify notation, we ignore parameters for actions. Similar actions with different parameters are treated as differing actions. For example, "run tftp with source directory `/home/tftp`" and "run tftp with source directory `/tftpboot`" are separate and distinct actions for our purposes.

## 3 Sequences of Actions

Given $P$ a set of actions, we form the set of all sequences of actions $\mathcal{F}(P)$ given by

$$\mathcal{F}(P) = \{\overline{p} = p_1 \circ p_2 \circ \cdots \circ p_{k-1} \circ p_k \mid k > 0, p_i \in P, 1 \le i \le k\} \cup \{\varepsilon\} \qquad (1)$$

where $\varepsilon$ is the *identity action* that consists of doing nothing at all. This is the *free semigroup with identity* over the alphabet $P$. Clearly $\mathcal{F}(P)$ is an infinite set, even though $P$ is finite.

The free semigroup $\mathcal{F}(P)$ consists of all possible sequences of actions. Many beliefs about the nature of actions in $P$ can be expressed as equivalence relations $\approx_i$ on $\mathcal{F}(P)$, where $\approx_i$ is a set of pairs $(\overline{p}, \overline{q})$ where $\overline{p}, \overline{q} \in \mathcal{F}(P)$. As usual, equivalence relations are reflexive ($(\overline{x}, \overline{x}) \in \approx_i$ for all $\overline{x} \in \mathcal{F}(P)$), symmetric ($(\overline{x}, \overline{y}) \in \approx_i \rightarrow (\overline{y}, \overline{x}) \in \approx_i$), and transitive ($(\overline{x}, \overline{y}), (\overline{y}, \overline{z}) \in \approx_i \rightarrow (\overline{x}, \overline{z}) \in \approx_i$ ).

In our physical interpretation of actions acting upon systems, two sequences of actions are equivalent if they accomplish the exact same result.

**Definition 1.** *Two configuration actions are equivalent if they achieve the same effect and are interchangeable in all contexts.*

In abstract terms, this means that equivalent actions are interchangeable within a sequence of actions:

**Definition 2.** *An equivalence relation $\approx$ is an* action equivalence *on $\mathcal{F}(P)$ if for all $\overline{x}, \overline{y}, \overline{a}, \overline{b} \in \mathcal{F}(P)$, $\overline{a} \approx \overline{b} \leftrightarrow \overline{x} \circ \overline{a} \circ \overline{y} \approx \overline{x} \circ \overline{b} \circ \overline{y}$.*

In other words, the substitution principle works. For the rest of this paper, we assume that all equivalences on sequences of actions are action equivalences according to this definition.

## 4 Convergent Actions

One concept of equivalence is that the actions are all convergent under the accepted definition of convergence[3, 10, 12]. An action $p$ is *idempotent* with respect to an action equivalence $\approx$ if repeating the action has the exact same effect as doing it once, i.e., $p \circ p \approx p$. A set of actions $P$ is *idempotent* if each individual action $p \in P$ is idempotent.

**Definition 3.** *We say a set of actions $P$ is* weakly convergent *if $P$ is idempotent as a set.*

This is the definition of convergence that controls actions in many common configuration management tools. In physical terms, an action $p$ is idempotent if repeating the action has no effect. Our algebraic definition of convergence does not include discussion of other attributes of convergence, such as non-intrusiveness. Non-intrusiveness is not an algebraic property, just an implementation detail.

In this paper, we will commonly study classes of actions by constructing an action equivalence relation that describes their properties.

**Definition 4.** *A weakly convergent set of actions $P$ (Definition 3) determines an action equivalence relation (Definition 2) $\approx_1$ on elements $\overline{p}, \overline{q} \in \mathcal{F}(P)$.*

According to this definition, $\overline{p} \approx_1 \overline{q}$ whenever $\overline{p}$ and $\overline{q}$ are sequences of actions that differ only in the number of times they repeat specific elements in order. For $p_1, p_2, ..., p_k \in P$ and a set of integer powers $\{n_i \mid n_i \geq 1, 1 \leq i \leq k\}$, the composition $p_1^{n_1} \circ p_2^{n_2} \circ \cdots \circ p_k^{n_k} \in \mathcal{F}(P)$ is equivalent under $\approx_1$ to any other sequence with differing choices of powers $n_i > 0$.

Unfortunately, the definition of weak convergence in Definition 3 has an undesirable property. The composition of idempotent actions need not be idempotent itself:

**Proposition 1.** *There are idempotent actions $p$ and $q$ where $q \circ p$ is not idempotent as a composite action.*

*Proof.* Idempotence of an action means that there is no difference in the *resulting state* of the system after repeating the action. As a very simple counterexample, suppose that there are two configuration state parameters $x$ and $y$, where the initial or "baseline" state of the configuration is that $x = y = 0$. Let $p$ be the action "if ($x$==1) then $y$=2", and let $q$ be "$x$=1". Clearly repeating either $p$ or

$q$ has no effect, so they are idempotent in isolation. Also, $q \circ p$ just sets $x$ to one (remembering that actions are applied from right to left), but $(q \circ p) \circ (q \circ p)$ sets $y$ to 2 as well. Clearly $q \circ p$ is not idempotent as an action if we start from the baseline state $x = y = 0$. If, e.g., one sets $x$ and $y$ initially to 1, rather than 0, $q \circ p$ *is* idempotent.

This proof illustrates the general principle that idempotence and convergence of operations is only meaningful relative to a choice of *baseline state* of a system: the state of the system before configuration actions begin. In this paper, we will presume that the baseline state is appropriately limited so that idempotence is present, and defer study of the structure of the baseline state for later work.

The reason that a composition of actions is not idempotent is that in the proof above, one action $p$ is *stateful*, so that *when* it is called determines what it does. There is a stronger characterization of what it means to be truly convergent that assures that compositions of actions are always idempotent.

**Definition 5.** *A set of actions $P$ is* pairwise stateless *with respect to an action equivalence relation $\approx_i$ if for any two actions $p, q \in P$, $p \circ q \circ p \approx_i p \circ q$.*

In other words, repeating $p$ *before* $p \circ q$ (remembering that we are evaluating actions from right to left) has no effect.

**Definition 6.** *A set of actions $P$ is* stateless *with respect to an action equivalence relation $\approx_i$ if for any action $p \in P$ and any sequence of actions $\overline{q} \in \mathcal{F}(P)$, $p \circ \overline{q} \circ p \approx_i p \circ \overline{q}$.*

**Lemma 1.** *A set of actions $P$ is stateless with respect to an action equivalence relation $\approx$ iff it is pairwise stateless with respect to the same relation.*

*Proof.* If $P$ is stateless, it is clearly pairwise stateless. So assume that $P$ is instead pairwise stateless and proceed by induction on the length of $\overline{q} \in \mathcal{F}(P)$. If $q$ has length 1, then it consists of one element $x \in P$ and

$$\begin{aligned} p \circ \overline{q} \circ p &= p \circ x \circ p && \text{hypothesis} \\ &\approx_i p \circ x && \text{pairwise statelessness} \\ &= p \circ \overline{q} && \text{hypothesis} \end{aligned} \tag{2}$$

If we assume that the lemma is true for compositions $\overline{q}$ containing $n$ components, then for $\overline{q} = q_1 \circ \cdots \circ q_n \circ q_{n+1}$ containing $n + 1$ components,

$$\begin{aligned} p \circ \overline{q} \circ p &= p \circ q_1 \circ \cdots \circ q_n \circ q_{n+1} \circ p && \text{definition} \\ &= (p \circ q_1 \circ \cdots \circ q_n) \circ q_{n+1} \circ p && \text{associativity} \\ &\approx_i (p \circ q_1 \circ \cdots \circ q_n \circ p) \circ q_{n+1} \circ p && \text{hypothesis} \\ &= p \circ q_1 \circ \cdots \circ q_n \circ (p \circ q_{n+1} \circ p) && \text{associativity} \\ &\approx_i p \circ q_1 \circ \cdots \circ q_n \circ (p \circ q_{n+1}) && \text{pairwise statelessness} \\ &= (p \circ q_1 \circ \cdots \circ q_n \circ p) \circ q_{n+1} && \text{associativity} \\ &\approx_i (p \circ q_1 \circ \cdots \circ q_n) \circ q_{n+1} && \text{hypothesis} \\ &= p \circ q_1 \circ \cdots \circ q_n \circ q_{n+1} && \text{associativity} \\ &= p \circ \overline{q} && \text{definition} \end{aligned} \tag{3}$$

so that $p \circ \overline{q} \circ p \approx_i p \circ \overline{q}$ and we are done.

**Definition 7.** *A set of actions $P$ is* convergent *with respect to an action equivalence relation $\approx$ if it is both idempotent and stateless with respect to that relation.*

Note that if $P$ contains the empty (identity) action $\varepsilon$, then statelessness straightforwardly implies idempotence: $p \circ p = p \circ \varepsilon \circ p \approx p \circ \varepsilon = p$.

Statelessness can be an elusive property in practice. The conditional `chmod` command "`chmod u+X file`" is convergent but not stateless. The "`u+X`" flag sets the owner execute flag if any execute flag is set for owner, group, or all. Interspersing a command that changes other execute flags, e.g., "`chmod g+x file`" or "`chmod a-x file`", can change the effect of a second instance of "`u+X`". The command "`chmod 755 file`" is by contrast both convergent and stateless.

## 5 Equivalent Actions

We can now define what it means for stateless actions to be equivalent.

**Definition 8.** *For any set of actions $P$, let $\approx_2$ represent the set of equivalences required by Definitions 2 and 7.*

In particular, $\overline{p} \approx_2 \overline{q}$ if there is a sequence of substitutions according to the definition that will transform $\overline{p}$ into $\overline{q}$.

Definition 7 makes the situation in Proposition 1 impossible. For $p, q \in P$, $q \circ p \circ q \circ p \approx_2 (q \circ p \circ q) \circ p \approx_2 (q \circ p) \circ p \approx_2 q \circ (p \circ p) \approx_2 q \circ p$.

For any set $S$ and equivalence relation $\approx$, the *factor set $S/\approx$* of the set and the relation is a partition of the set into pairwise disjoint subsets $S_i$, each of whose elements are equivalent according to the equivalence relation. For a set of actions $P$, consider the factor set $\mathcal{F}(P)/\approx_2$. This is the set of *distinct actions* represented by $\mathcal{F}(P)$ under the equivalence $\approx_2$.

**Lemma 2.** *Every element in $\mathcal{F}(P)$ is equivalent under $\approx_2$ to one containing only the leftmost instance of each duplicated action. Thus $\mathcal{F}(P)/\approx_2$ consists of a finite number of subsets.*

*Proof.* Let $x \in \mathcal{F}(P)$. The goal is to express the word $x$ as another equivalent word $x'$ possessing only the leftmost instance of each distinct action from $x$. If $x$ consists of one action $p \in P$, let $x' = x$ and we are done. So presume that the lemma is true for strings of $n$ objects and suppose that $x$ consists of $n+1$ objects $p_1 \circ \cdots \circ p_{n+1}$. Then $y = p_1 \circ \cdots \circ p_n$ consists of $n$ actions and the lemma applies, so that there is a sequence $y'$ equivalent with $y$ and containing only the leftmost instance of each action in $y$. Then $x \approx_2 y \circ q_{n+1} \approx_2 y' \circ q_{n+1}$. Now if $q_{n+1}$ is not identical to some element of $y'$, $y' \circ q_{n+1}$ consists of unique elements and is an appropriate choice for $x'$. If it is identical to some $q_i$ that is a term of $y'$, then by Definition 7, $x \approx_2 y' \circ q_{n+1} \approx_2 y'$ and $y'$ is an appropriate $x'$.

The purport of this proof is that even though the free semigroup $\mathcal{F}(P)$ is infinite, we only "care about" a finite factor set of that infinite set.

**Lemma 3.** *Every element $\overline{p}$ of $\mathcal{F}(P)$ is* idempotent *with respect to $\circ$ and $\approx_2$, i.e., $\forall \overline{p} \in \mathcal{F}(P), \overline{p} \circ \overline{p} \approx_2 \overline{p}$.*

*Proof.* Given $\overline{p} \in \mathcal{F}(P)$, apply Lemma 2 to create an equivalent composition $\overline{p}' \approx_2 \overline{p}$ that contains no duplicates. Then

$$\begin{aligned} \overline{p} \circ \overline{p} &\approx_2 \overline{p}' \circ \overline{p}' && \text{Definition 1} \\ &\approx_2 \overline{p}' && \text{Lemma 2} \\ &\approx_2 \overline{p} && \text{construction} \end{aligned} \tag{4}$$

and $\overline{p}$ is idempotent with respect to $\approx_2$.

**Lemma 4.** *If we define for $\widetilde{p}, \widetilde{q} \in \mathcal{F}(P)/\approx_2$, $\widetilde{p} \mathbin{\widetilde{\circ}} \widetilde{q}$ as the unique $\widetilde{r} \in \mathcal{F}(P)/\approx_2$ such that*

$$\{\overline{a} \circ \overline{b} \mid \overline{a} \in \widetilde{p}, \overline{b} \in \widetilde{q}\} \subseteq \widetilde{r} \tag{5}$$

*then $\widetilde{\circ}$ is well defined and $(\mathcal{F}(P)/\approx_2, \widetilde{\circ})$ is a semigroup.*

*Proof.* The nature of equivalence is agreement on order of first appearance of each action. By Lemma 2, $\widetilde{p}$ and $\widetilde{q}$ have unique elements $\overline{p} \in \widetilde{p}$ and $\overline{q} \in \widetilde{q}$ with minimal length, and the sequence $\overline{p} \circ \overline{q}$ can be transformed by Lemma 2 to a similar representative element $\overline{r}$ of $\widetilde{r}$. The choice of $\overline{r}$ uniquely determines the contents of $\widetilde{r}$ by Lemma 2 and $\widetilde{\circ}$ is thus well-defined.

Note that for $\overline{p} \in \widetilde{p} \in \mathcal{F}(P)/\approx_2$ and $\overline{q} \in \widetilde{q} \in \mathcal{F}(P) \approx_2$, $\overline{p} \approx_2 \overline{q}$ exactly when $\widetilde{p} = \widetilde{q}$.

A semigroup whose elements are all idempotents is called a *band*. We have thus shown that:

**Theorem 1.** *If $P$ is a finite set of convergent actions obeying the equivalence $\approx_2$ and $\widetilde{\circ}$ is the operation defined in Equation 5, $(\mathcal{F}(P)/\approx_2, \widetilde{\circ})$ is a finite band.*

Note that this is not true in general if equivalence is expressed instead according to Definition 3. We have already shown that there are primitive actions $p, q$ for which $p \circ q$ is not idempotent. Not only is $\mathcal{F}(P)/\approx_1$ not idempotent; *it may not be a semigroup at all.* It is only a semigroup if one can show that for some operation $\overline{\circ}$ and every $\widetilde{p}, \widetilde{q} \in \mathcal{F}(P)/\approx_1$, $\widetilde{p} \mathbin{\widetilde{\circ}} \widetilde{q} \subseteq \widetilde{r}$ for some uniquely determined $\widetilde{r} \in \mathcal{F}(P)/\approx_1$. This is possible but depends on the nature of particular actions in $P$.

## 6 Value of Idempotence

The properties and possible structures of idempotent semigroups have been very extensively studied[20–22]. As a summary, we quote the following well-known results without proof.

A *subsemigroup* $(R, \circ)$ of a semigroup $(S, \circ)$ is a subset $R \subseteq S$ closed under $\circ$, i.e., for $r_1, r_2 \in R$, $r_1 \circ r_2 \in R$. A *commutative band* $(S, \cdot)$ is one in which all actions commute, i.e., for $x, y \in S$, $x \cdot y = y \cdot x$. A band $Q$ is a *matrix band* if it is isomorphic to some band $(\Gamma \times \Delta, \cdot)$ where $\Gamma$ and $\Delta$ are (arbitrary) disjoint alphabets, and for $(x, y), (z, w) \in \Gamma \times \Delta$, $(x, y) \cdot (z, w) = (x, w)$. In other words, a matrix band is a particularly simple kind of band. Note in particular that elements of a matrix band never commute except with themselves, because $(x, y) \cdot (z, w) = (x, w)$ while $(z, w) \cdot (x, y) = (z, y)$.

**Theorem 2.** *A semigroup all of whose elements are idempotents is a commutative band of matrix bands of unit groups ([22], Corollary 2.14, page 320).*

The proof of this theorem is too long to include, but the meaning for configuration management tools is straightforward. A configuration management tool implements a set of actions $P$, where we have shown that under the definition of convergence we consider correct, $\mathcal{F}(P)/\approx_2$ is a finite band. Its actions can be partitioned into a set of disjoint non-commutative subsets $\mathcal{B}_1, \ldots, \mathcal{B}_n$ that, considered as elements of a larger semigroup, commute. In this semigroup, the product of two elements $\mathcal{B}_i$ and $\mathcal{B}_j$ is that unique semigroup $\mathcal{B}_k$ containing all products of pairs $b_i \in \mathcal{B}_i$ and $b_j \in \mathcal{B}_j$. Each member $\mathcal{B}_i$ of the larger commutative band represents all appropriate values for one parameter, and can be partitioned further into subsets that form a semigroup of non-commutative members $\mathcal{B}_{i1}, \ldots, \mathcal{B}_{ik}$ representing particular values for a parameter.

We have thus proven that configuration parameters exist in all cases, without assuming that they do. Existence of parameters is an *algebraic* property that arises from assumptions of action equivalence, idempotence, and statelessness!

## 7 Meaning of Commutativity

Commutivity or non-commutivity of idempotent actions has a rather simple meaning.

**Definition 9.** *Two idempotent actions $a, b$ are* consistent*[12] iff they commute, i.e., $a \circ b = b \circ a$.*

Consistent actions *agree in intent* on how to arrange a configuration; inconsistent actions do not. Note that *orthogonal* actions that affect differing parts of the system are automatically consistent as well.

To understand this in a simple case, consider each action to be operating on an extremely large vector of bits called "the configuration". Every action asserts some fixed values for a subset of bits of the configuration. Actions that agree on any common values commute, including actions that act on different parts of the configuration and thus cannot conflict. Conflicting actions do not commute.

As a concrete example, suppose that configuration consists of five bits $a_1 \ldots a_5$. A configuration is then a set of values for all five, i.e., a mapping from $\{a_i\} \rightarrow \{0, 1\}$. Configuration actions are then assertions of values of particular integers. If configuration action $B = \{a_1 := 1; a_5 := 0\}$ , this commutes with $C = \{a_1 := 1; a_2 := 1\}$ but not with either $D = \{a_1 := 0; a_2 := 0\}$ or $E = \{a_5 := 1\}$.

## 8 Implications for Configuration Management Tools

We have shown so far that particularly nice properties are exhibited by any set of configuration primitives possessing statelessness as described in Definition 7. Unfortunately, few actions in current configuration management tools are stateless.

## 8.1 Convergence and File Editing

The premier tool for convergent system administration is Cfengine[3, 4]. While Cfengine's abilities to copy files, replicate links, and change protections are fully convergent in the above sense, its extensive facilities for editing files are not convergent according to the above algebraic definition. Editing operations commonly conflict in intent, thus rendering them both non-commutative and problematic.

For example, if one reorders the Cfengine editing actions:

```
editfiles:
  all::
    { /etc/services
      deleteLinesMatching "tftp"
      appendIfNotPresent "tftp 6900/udp"
    }
```

(that have obvious meanings) there will be *no* line describing tftp instead of the *new* line describing tftp. File editing is procedural, not declarative; the action of deleting an old configuration line before adding a new one is non-commutative.

## 8.2 Rethinking File Editing

The argument above suggests that the true mathematical language of convergent system administration consists of self-consistent *assertions of state*, where sets of non-conflicting and consistent assertions *commute*. These assertions are *not* similar to editing commands; they assert intent without the implicit changes of intent that plague any attempt to edit during reconfiguration.

To address this problem, we completely rethought and redesigned the process of file editing to be declarative rather than procedural. Each file is treated like a database, and editing commands become precise assertions describing contents in the file. Rather than the above Cfengine editing script, one might write the stateless assertion:

```
in /etc/services
  where field1=tftp
    line is "tftp 6900/udp"
```

or perhaps:

```
in /etc/services
  where field1=tftp
    make field2=6900/udp
```

or even (splitting up fields and renaming):

```
in /etc/services
  where service=tftp
    make port=6900 proto=udp
```

To have the effect of deleting a line, one asserts its absence:

```
in /etc/services
  where service=tftp
    omit
```

This is clearly more trouble than a simple edit; one must translate from imperative language to declarative. The interpreter of the declaration must understand the syntax of the file.

Adapting this idea to configuration files that are not linear lists is non-trivial but feasible. One must decompose each file into regions describing particular intents, and parse each separately. As files typically follow a natural hierarchy, the assertion language naturally follows that structure. For example, to configure a virtual server, one might write

```
in /etc/httpd.conf
  with virtual server foo.bar.com
    make port=9001
```

Accomplishing this level of declarative interaction with configuration is not a single-layer process. There are several layers of interaction involved, from low to high:

1. A *syntactic* layer that describes the syntax of each configuration file, i.e., a parser.
2. A *constraint* layer that describes which configuration states are meaningful.
3. A *policy* layer that describes which configuration states are allowed by site policy.
4. An *intent* layer that converses through the constraint and syntactic layers to create a particular behavioral effect.
5. A *validation* layer that insures that configuration does indeed change behavior.

A particular declaration, such as the examples above, travels downward (backward) through these layers to achieve a particular intent.

Some layers are generic; the syntax of particular configuration files is portable among all operating systems, as is the process of validating effects. For the most part, intent is also portable, but there are two kinds of constraints: those imposed by the system being configured and those configured by humans. Most of this infrastructure is thus reusable, with the exception of the layer in the *middle* of the sandwich that describes site policies. Creating the layers is significant work, but most of it must be done once and is reusable everywhere.

## 8.3 Generative Management

A *generative configuration management tool* expresses actions in terms of the complete replacement of configuration by new configuration. A typical tool replaces all files at each configuration step, so that each configuration action expresses a state for every configuration file. The simplest generative model is “wipe the disk, re-install, and regenerate all configuration files", as in MIT's Project Athena. More complex generative models such as LCFG[1, 2], PSGCONF[24], database-driven models[18, 19], and some flavors of Arusha[14, 15] selectively replace all configuration files while leaving all else alone.

A generative configuration tool decomposes configuration into generative actions $g \in G$ each of which produces and asserts the contents of an entire configuration. For any generative actions $g$ and $h$, $h \circ g = h$, i.e., the last action wins. This means that each element of a generative semigroup is a "left zero" of the semigroup, so that the structure is "left annihilating". These $h$'s are the $b_i$'s in the above discussion.

One disadvantage of generative configuration management is the expense of building the initial model, the "problem of semantic distance"[11]. Typical configuration procedures are documented for humans in terms of scripts, not in terms of assertions. Moving that configuration from the script to a generator means turning scripts into assertions about their effects, a problem that is as difficult as proving a script to be correct. The generator incorporates distributed knowledge from multiple scripts into an expression of global knowledge, consistency, and precedences.

### 8.4 Imperative Management

By contrast, managing a system by imperative methods is the most complex method algebraically. The actions in this method are imperative scripts, so that interactions between scripts can potentially be arbitrarily complex. Unfortunately, scripts are also necessary. As a legacy, most systems are initially built through scripts. In general, however, concatenating two scripts does not guarantee a composite effect of both scripts.

Scripts do have several advantages. They are the legacy way to accomplish change and are already part of tasks such as package installation. They work well in the absence of changes. They establish a baseline within which another tool can work, and that is difficult to establish otherwise.

## 9 Conclusions

We propose a new and more strict notion of convergence that rules out common practices previously considered to be convergent. Using this definition alone, the properties of a typical configuration management system emerge from the theory of semigroups. Using the prior definition, no such properties emerge, and the resulting inelegance of the system leads to usability problems in crafting strongly convergent actions from weakly convergent primitives.

In a simple algebraic sense, imperative, generative, and convergent methods form a hierarchy in which imperative methods are closest to the machine and convergent methods closest to becoming self-healing. During initial machine bootstrapping, imperative methods are inescapable; the cost of replacing them with generative or convergent methods is too high. After initial configuration, convergent methods have the potential to inadequately cover all required options, a behavior not exhibited by typical generative systems, The initial run of a convergent system must "configure all variables" and thus has a generative flavor.

So, we conclude from this study that an ideal configuration management tool actually has attributes of each of the strategies:

1. An imperative bootstrap.
2. A generative initial configuration.
3. A convergent ongoing management process.

where the latter is convergent in the strong sense (Definition 7) rather than the weak sense exhibited in most current convergent tools, including Cfengine.

We strive for order in a disorderly world. Strongly emergent properties are the result of strong axioms and an uncompromising adherence. The cost of this adherence is sometimes high. We feel that the value exceeds the cost, and that future self-healing tools will adopt strongly convergent methods to assure the algebraic elegance that naturally results in simplicity of use.

## 10 Acknowledgements

This work is not the product of individuals, but of a coordinated inquiry among a large and diverse intellectual community. We are grateful to the Large-Scale System Configuration mailing list (lssconf) for providing sustained discussion of the strategies for configuration management and their impact. George Leger and David Krumme both provided detailed, in-depth comments and corrections to the mathematics. Special thanks to Mark Burgess, Paul Anderson, and Steve Traugott, for providing the main fuel for this debate with their groundbreaking work. This work was supported in part by an equipment grant from Network Appliance Corporation.

## References

1. P. Anderson, "Towards a High-Level Machine Configuration System" *Proc. LISA-VIII*, USENIX Assoc., 1994.
2. P. Anderson, P. Goldsack, and J. Patterson, "SmartFrog Meets LCFG: Autonomous Reconfiguration with Central Policy Control", to appear in *Proc. LISA-XVII*, USENIX Assoc., San Diego, CA, 2003.
3. M. Burgess, "A Site Configuration Engine", *Computing Systems* **8**, 1995.
4. M. Burgess and R. Ralston, "Distributed Resource Administration Using Cfengine", *Software: practice and experience* **27**, 1997.
5. M. Burgess, "Computer Immunology", *Proc. LISA-XII*, Boston MA, USENIX Assoc., 1998.
6. M. Burgess, "Theoretical System Administration", *Proc. LISA-XIV*, New Orleans LA, USENIX Assoc., 2000.
7. Lionel Cons and Piotr Poznanski, "Pan: A High-Level Configuration Language", *Proc. LISA-XVI*, USENIX Assoc., Philadelphia, PA, 2002.

8. A. Couch, “SLINK: simple, effective filesystem maintenance abstractions for community-based administration”, *Proc. Lisa-X*, USENIX Assoc, 1996.
9. A. Couch, “Chaos out of order: a simple, scalable file distribution facility for ‘intentionally heterogeneous’ networks”, *Proc. LISA-XI*, USENIX Assoc., 1997.
10. A. Couch and M. Gilfix, “It’s elementary, dear Watson: applying logic programming to convergent system management processes”, *Proc. Lisa-XIII*, USENIX Assoc., 1999.
11. Alva L. Couch, “An expectant chat about script maturity”, *Proc. LISA-XIV*, USENIX Assoc., 2000.
12. Alva L. Couch and Noah Daniels, “The maelstrom: network service debugging via ‘ineffective procedures’ ”, *Proc. LISA-XV*, USENIX Assoc., 2001
13. A. Couch, J. Hart, E. Greenlee, and D. Kallas, “Seeking Closure in an Open World: A Behavioral Agent Approach to Configuration Management”, to appear in *Proc. LISA XVII*, USENIX Assoc., San Diego CA, 2003.
14. Matt Holgate and Will Partain, “The Arusha Project: A framework for collaborative Unix system administration”, *Proc. LISA XV*, USENIX Assoc., San Diego CA, 2001.
15. Matt Holgate, Will Partain, et al, “The Arusha Project Web Site”, http://ark.sourceforge.net.
16. L. Kanies, “Practical and Theoretical Experience with ISconf and Cfengine”, to appear in *Proc. LISA XVII*, USENIX Assoc., San Diego CA, 2003.
17. Frode Eika Sandnes, “Scheduling partially ordered events in a randomised framework - empirical results and implications for automatic configuration management”, *Proc. LISA XV*, USENIX Assoc., San Diego CA, 2001.
18. Jon Finke, “An improved approach for generating configuration files from a database”, *Proc. LISA-XIV*, USENIX Assoc., 2000.
19. , J. Finke, “Generating Configuration Files: The Director’s Cut”, to appear in *Proc. LISA-XVII*, USENIX Assoc., San Diego, CA, 2003.
20. P. A. Grillet, *Semigroups: An Introduction to the Structure Theory*, Marcel Dekker, Inc, New York, NY, 1995.
21. J. M. Howie, *An Introduction to Semigroup Theory*, Academic Press, 1976.
22. E. S. Ljapin, *Semigroups*, American Mathematical Society, Providence, RI, 1963.
23. Mark Logan, Matthias Felleisen, and David Blank-Edelman, “Environmental Acquisition in Network Management” *Proc. LISA XVI*, USENIX Assoc., Philadelphia, PA, 2002.
24. M. D. Roth, “Preventing Wheel Reinvention: The Psgconf System Configuration Framework”, to appear in *Proc. LISA-XVII*, USENIX Assoc., San Diego, CA, 2003.
25. Steve Traugott and Joel Huddleston “Bootstrapping an Infrastructure”, *Proc LISA XII*, USENIX Assoc., Boston, MA 1998.
26. Steve Traugott and Lance Brown, “Why order matters: Turing equivalence in automated systems administration” *Proc. LISA XVI*, USENIX Assoc., Philadelphia, PA, 2002.