On system rollback and totalised fields

An algebraic approach to system change

Mark Burgess and Alva Couch

20th June 2011

This paper is dedicated to Jan Bergstra on the occasion of his 60th birthday.

Abstract

In system operations the term rollback is often used to imply that arbitrary changes can be reversed i.e. 'rolled back' from an erroneous state to a previously known acceptable state. We show that this assumption is flawed and discuss error-correction schemes based on absolute rather than relative change.

Insight may be gained by relating change management to the theory of computation. To this end, we reformulate previously-defined 'convergent change operators' of Burgess into the language of groups and rings. We show that, in this form, the problem of rollback from a convergent operation becomes equivalent to that of 'division by zero' in computation. Hence, we discuss how recent work by Bergstra and Tucker on zero-totalised fields helps to clear up long-standing confusion about the options for 'rollback' in change management.

1 Introduction

The assumption that it is possible to reverse changes, or create generic 'undo' buttons in arbitrary software systems, remains a persistent myth amongst software developers, system designers, and system administrators in all areas of computing. The term 'rollback' is often used to describe this form of repair, usurped from its original usage in database transaction theory[1, 2]. In current usage, rollback refers to the act of undoing what has been done; it is intimately related to checkpointing[3, 4, 5, 6, 7], version control and release management.

In single-threaded and parallel software applications, many authors have developed a 'journaling' approach to reversibility and rollback (see foregoing references on checkpointing). A stack of state-history can be kept to arbitrary accuracy (and at proportional cost), provided there is sufficient memory to document changes. In more general 'open' (or incompletely specified) systems the cost of maintaining history increases without bound as system complexity increases. We shall show that arbitrary choices – which we refer to as *policy decisions* – are required to choose remedies for incomplete specifications.

A model of change that includes the idea of maintenance of an absolute intended state was introduced in [8, 9]. This model has been realized in the software Cfengine[10], and was further elaborated upon using an alternative formulation in [11]. The crux of this approach is to bring about a certainty of outcome, even in an incompletely specified (or so-called 'open') system, and has proved to have several advantages over traditional relativistic approaches to change, including that it allows autonomic repair of developing problems. However, this certainty is brought at the expense of a loss of history that would enable the reversal of certain kinds of changes.

In this paper we discuss a formulation of policy-based change management from an unorthodox perspective: that of computation with data-types. In particular, we note the relationships to recent work by Bergstra and Tucker on division-safe calculation in algebraic computation[12, 13]. We show that reversibility in system management and totalization of rational fields are, in fact, closely related.

The discussion is potentially large, so we set modest goals. We begin by reviewing basic ideas about reversibility, and then recall the notion of 'convergent' or 'desired-state' operations. We explain the relationship of these abstract operators to the zeros of rings and fields and we show how the inverse zero operation 0^{-1} can be viewed as an attempt to 'roll back' state from such a convergent operation, in one interpretation of configuration management. This makes a connection between 'calculation' and system configuration, implicit in the encoding of data into types. Finally, noting that zero plays two roles for $+, \cdot$ in ring computation, we compare the remedies for division-safe calculation with options for reversal in change management.

2 Notation

We follow the notation of [8] in writing a generic operators as letters with carets over them, e.g. $\hat{O}_1, \hat{O}_2$, etc, while non careted operators are specific representations, usually matrices, e.g. Δ, μ, C. Generic states on which these operators act are written in Dirac notation $|q\rangle$, with its distinctive typography that distinguishes it from particular representations, and q here is a label that selects a particular state from the available set. Operators without carets are assumed to be specific realizations, usually matrices or vectors. A resulting state after applying an operator $\hat{O}_1$ to a system in the state $|q\rangle$ is written as $\hat{O}_1|q\rangle$. The symbol t will represent a time, and δt is a time increment. Similarly δX will imply a relative change in quantity X. S will denote a general set, R a ring and F a field. G is a group with elements $g_1, g_1^{-1}, \ldots, I$, where I is the identity element. When discussing rings and fields and division-safe calculation, we shall stay close to the notation of Bergstra and Tucker[12, 13].

3 Conceptual overview

Given that our goal is to bridge several somewhat-disparate scientific cultures, let us begin by painting the larger picture as we see it. We ask the forebearance of readers as we attempt to straddle unfamiliar disciplines to give this account. Our train of thought is as follows:

- We model system configurations as data types, including numbers and strings in such a way as to allow algebraic structures that are 'isomorphic representations' of state spaces. This work was originally formulated for a different audience, and in a different language, in order to define the concept of maintenance of high level systems[8, 14].

- The algebra of the configurations can therefore be modelled by the algebra of the representations. In particular groups, rings and fields come into play, which emphasizes that changes may be viewed as computations[9, 15].

- The familiar structure of rings and fields bring a great deal of algebraic knowledge and methodology, including matrix algebra, and recent work on totalisation of fields[12, 13, 16].

- The less orthodox matters of division by zero in totalised fields enter when attempting to define reversibility and error correction in this high-level model of change, by reversal by policy. This offers some guidance for resolving a problem of incomplete information in a framework of absolute change.

The associations we introduce have interesting things to say about error correction. We begin by explaining the issues of change management by making an identification between change and computation, using rings and fields. We then show how an operator formalism can be used to track changes of state, before moving to discuss the inverse of such change operations.

4 Modelling configuration parameters

Configuration management is largely viewed as a process of setting and maintaining the values of *configuration parameters* that control or influence software behavior. A parameter is usually a number or string having a finite (though potentially large) set of useful values. For example, one parameter might be the number of threads to use in a web server, with a typical value of 10. Another might be a 'yes' or 'no' string determining whether a web server should be started at boot time.

Although this view of a computer is often presented as being orthogonal to matters of computation, there are plenty of reasons to remove any such distinction. There are, indeed, multiple senses in which any machine or even physical process may be considered to participate in a computation as it evolves in time from some initial state. Indeed, the construction of computation from automata theory demonstrates this. For the purposes of this work, we find it useful to view arbitrary changes of state as computations for two reasons:

1. Because numerical descriptions of dynamical systems abound in the natural sciences, and there is a wealth of cross-disciplinary techniques based on rings and fields that can be employed to model them in a convenient framework.
2. Because this allows us to make contact with an important argument concerning inverses and the totality of functions based on rational numbers.

Ultimately the desire for alternative descriptions of computer behaviour has to do with understanding the trends and statistical behaviours inherent in non-deterministic systems.

Given that configuration parameters can be viewed as numbers and strings, we propose a model for these bsaed on a field structure for each configuration parameter X by injectively mapping its possible values (as a set G_X) to a subset of some field $(F_X, +, \cdot)$, by an injection $\phi : G_X \to F_X$. There are three possible structures for G_X, including sets of rational numbers, finite sets of numbers, and sets of strings. If $G_X = \mathbb{Q}$ is the set of rational numbers, G_X maps to itself. Finite sets of integers $G_X \subset \mathbb{Z}$ containing n possible values can be mapped to the first n integers in $\mathbb{Q}$, starting from 0. String parameter sets containing a finite number of values can be likewise mapped to the first n integers in $\mathbb{Q}$. For example, a string parameter taking the values 'yes' and 'no' might be mapped via:

$$\begin{aligned} \text{'yes'} &\mapsto 1 \\ \text{'no'} &\mapsto 0 \end{aligned} \tag{1}$$

The purpose of $\phi : G_X \to \mathbb{Q}$ is to impart meaning to the field operations $+$ and $\cdot$ for parameter values in G_X. We may extend G_X to a set G'_X by adding potentially meaningless values, and extend ϕ to a bijection $\phi' : G'_X \to F_X$. The exact structure of this mapping does not matter. We may thus define

$$\begin{aligned} q + r &\equiv \phi'^{-1}(\phi'(q) + \phi'(r)) \\ q \cdot r &\equiv \phi'^{-1}(\phi'(q) \cdot \phi'(r)) \end{aligned} \tag{2}$$

whenever ϕ'^{-1} exists. Since F_X is a field, G'_X satisfy the usual field signature:

$$\begin{array}{rcl}
\textbf{signature} & & \\
\textbf{sorts} & & G'_X \\
\textbf{constants} & & \\
0 & : & \to G'_X \\
1 & : & \to G'_X \\
\textbf{operations} & & \\
+ & : & G'_X \times G'_X \to G'_X, \\
- & : & G'_X \times G'_X \to G'_X, \\
\cdot & : & G'_X \times G'_X \to G'_X, \\
\textbf{end} & &
\end{array} \tag{3}$$

(4)

with properties:

$$\begin{array}{rrcl}
& \textbf{equations} & & \\
& x + 0 & = & 0 + x = x, \\
& 1 \cdot x & = & x \cdot 1 = x \\
(\forall x, -x \in G'_X) & x + (-x) & = & (-x) + x = 0 \\
(\forall x \in G'_X, x \neq 0, x^{-1} \in G'_X) & x \cdot x^{-1} & = & x^{-1} \cdot x = 1 \\
& x + y & = & y + x \\
& x \cdot y & = & y \cdot x \\
& (x \cdot y) \cdot z & = & x \cdot (y \cdot z) \\
& (x + y) + z & = & x + (y + z) \\
& (x + y) \cdot z & = & (x \cdot z) + (y \cdot z). \\
& \textbf{end} & &
\end{array} \tag{5}$$

The point of this introductory discussion is to be able to propose the following point of departure in the discussion[8]:

Proposition 1 *Without loss of generality, we may consider the range of values G_X of any configuration parameter X to have a field structure $(G'_X, +, \cdot)$ for some $G'_X \supset G_X$, where G'_X is isomorphic to a field $(F_X, +, \cdot)$.*

Usually, the structure of ϕ' is simple. For example, via the mapping in 1,

$$\begin{array}{rcl}
\text{'yes'} + \text{'no'} & = & \text{'yes'} \\
\text{'no'} + \text{'no'} & = & \text{'no'} \\
\text{'yes'} \cdot \text{'yes'} & = & \text{'yes'} \\
\text{'yes'} \cdot \text{'no'} & = & \text{'no'}
\end{array} \tag{6}$$

Thus 'yes' is the multiplicative unit and 'no' is the additive unit of G'_X, respectively.

In the rest of this paper, we will not consider the semantics of G'_X, so there is no need to distinguish between the base field $(F_X, +, \cdot)$ and its image $(G'_X, +, \cdot)$ in parameter space. We will use $(F_X, +, \cdot)$ to refer both to the base field and its isomorphic image in parameter space.

We now have a basis from which to develop possible multiple approaches to studying computer behaviour in non-programmatic, rule-based frameworks. Philosophically this sets us on a par with other areas of natural science where one begins by describing actual observed phenomena that may differ from the scope of programmatic expectations. From the low level view of symbolic change, we shall progress up the abstraction ladder to encompass arbitrary data types. In particular, to make contact with [8], we want to think of a calculus of state changes $|X\rangle \to |X + \delta X\rangle$ over an interval $t + \delta t$. To speak of a reversal of this change, we need also '$-$', thus we are led to groups. These $\pm$ translational operations are usually commutative and quickly lead us to the well-known concept of rings $R = (S, +, \cdot)$.

It is worth reminding readers that the symbolic form of the binary operators $+$ and $\cdot$ can always be written as a multiplication by a different kind of object by raising the dimensionality of the symbols by one, i.e. by going to a tuple formulation[17] (though the reverse is not true). Thus $X \to X + \delta X$ may be written

$$\vec{X} \to \Delta X \cdot \vec{X} \tag{7}$$

where ΔX is a matrix. A simple realization of this may be given in terms of linear functions or matrices as follows:

$$\Delta X = \begin{pmatrix} 1 & \delta X \\ 0 & 1 \end{pmatrix} \text{ and } \vec{X} = \begin{pmatrix} X \\ 1 \end{pmatrix} \tag{8}$$

Hence, at the expense of one additional point in an extra tuple dimension, we may write a group translation as a group multiplication and simplify notation to cover ring operations for a multiple data types in the manner of multiplicative forms. This is an elementary fact of representation theory which we shall use repeatedly in the following sections.

5 Modelling parameter changes

Viewing parameter values as a subset of a field (e.g., the rational numbers), opens up a more humanly familiar way of thinking about system configuration changes, with a corresponding intuitive algebraic structure in which one can make use of intuitions like 'increasing' and 'decreasing' to discuss machine behaviours over time. So, quite apart from the specific subject of system rollback, the manifesto of relating state and computation is of basic value to understanding actual computer behaviour in broader scientific terms. However, we shall focus mainly on the question of change and reversals here.

Technically, our transformations allow us to distinguish three approaches to change in the value of a parameter, making precise the notion of change $q \to q + \delta q$, used in [8]. We call the three approaches *relative* (Δ), convergent or *absolute* (C), and *multiplicative* (μ) or scale change, and we now wish to separate these, so as to distinguish their properties more clearly.

We partition the field algebra into partial functions using the trick from representation theory to write binary addition in the form of a parameterized unary group multiplication by introducing a tuple form with one extra dimension[17]. We write the parameter X as a vector $|X\rangle$:

$$|X\rangle = \begin{pmatrix} X \\ 1 \end{pmatrix} \tag{9}$$

and use standard matrix algebra to express changes, building on $+$ and $\cdot$ for elements of F_X.

- A *multiplicative change* in a parameter X is the result of a matrix operation of the form:

$$|X'\rangle = \mu(q)\,|X\rangle = |q \cdot X\rangle \tag{10}$$

where q is an element of the field $(F_X, +, \cdot)$ and $\mu(q)$ is defined as:

$$\mu(q) = \begin{pmatrix} q & 0 \\ 0 & 1 \end{pmatrix} \tag{11}$$

This has the effect of setting $X' = q \cdot X$, and semantically, is a scaling operation.

- An *absolute change* has the form:

$$|X'\rangle = C(q)\,|X\rangle = |q\rangle \tag{12}$$

where $C(q)$ is defined as

$$C(q) = \begin{pmatrix} 0 & q \\ 0 & 1 \end{pmatrix} \tag{13}$$

An absolute change is the equivalent of setting $X = q$ for some $q \in F_X$.

- A *relative change* in a parameter X takes the form $X' = X + \delta X$ where $+$ is the field addition operation for F_X and $\delta X \in F_X$. We can write a relative change as

$$|X'\rangle = \Delta(\delta X)\,|X\rangle = |X + \delta X\rangle \tag{14}$$

where $\Delta(q)$ is defined as

$$\Delta(q) = \begin{pmatrix} 1 & q \\ 0 & 1 \end{pmatrix} \tag{15}$$

Any relative change is a linear change. The converse is not true; the linear operators $\mu(x)$ and $C(x)$ are not equivalent to relative operators.

Composing combinations of $\mu(q)$, $C(q)$, and $\Delta(q)$ by matrix multiplication always results in a linear operator of the form:

$$\begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} \tag{16}$$

where a and b are elements of F_X.

Note that $\mu(F_X \setminus \{0\}) = \{\mu(q) \mid q \in F_X \setminus \{0\}\}$ is a (multiplicative) Abelian group, because $\mu(q)$ has multiplicative inverse $\mu(q^{-1})$ for $q \neq 0$. Likewise $\Delta(F_X) = \{\Delta(q) \mid q \in F_X\}$ is a (multiplicative) Abelian group, where $\Delta(q)$ has multiplicative inverse $\Delta(-q)$. $C(q)$, by contrast, is always singular and has no multiplicative inverse, so that $C(F_X) = \{C(q) \mid q \in F_X\}$ is not a group.

Also note that

$$\begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix}\begin{pmatrix} c & d \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} ac & ad + b \\ 0 & 1 \end{pmatrix} \tag{17}$$

while

$$\begin{pmatrix} c & d \\ 0 & 1 \end{pmatrix}\begin{pmatrix} a & b \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} ac & bc + d \\ 0 & 1 \end{pmatrix} \tag{18}$$

so multiplication of elements in the span of $\mu(F_X) \cup \Delta(F_X) \cup C(F_X)$ is only commutative if $ad + b = bc + d$.

Finally, note that the vectors $|q\rangle$ can still be thought of as a field with operations:

$$|q\rangle + |r\rangle = \begin{pmatrix} q \\ 1 \end{pmatrix} + \begin{pmatrix} r \\ 1 \end{pmatrix} \equiv \begin{pmatrix} q+r \\ 1 \end{pmatrix} = |q+r\rangle \tag{19}$$

and

$$|q\rangle \cdot |r\rangle = \begin{pmatrix} q \\ 1 \end{pmatrix} \cdot \begin{pmatrix} r \\ 1 \end{pmatrix} \equiv \begin{pmatrix} q \cdot r \\ 1 \end{pmatrix} = |q \cdot r\rangle \tag{20}$$

Thus it is reasonable to write things like $\delta q = |q_1\rangle - |q_2\rangle$ and $|q_2\rangle = |q_1\rangle + \delta q$. We will often write the latter as $|q_1 + \delta q\rangle$ without confusion, and will often switch between additive ($|q + \delta q\rangle$) and multiplicative ($\Delta(\delta q)\ |q\rangle$) representations of addition.

6 Modelling changes over time

So far we have a non-temporal model of change; however, since change operators $C(q)$ do not commute, partial orderings of operations are important and some independent time becomes the natural expression of sequence at our system level of abstraction. Using our framework we shall now say that between any two times t and $t + \delta t$, a system state $|q\rangle$ might change from $|q\rangle$ to $|q + \delta q\rangle$[8].

Let us digress slightly for a moment to explain time. Time can only be measured by changes of state; thus to mark time absolutely, one would need a standard external clock or reference change marker with sufficient resolution to calibrate system changes. However, current state is not usually archived when unintended changes occur, say as a result of environmental errors or influences, so clocks get muddled when there are overlapping sources of change. This is one way to see why rollback goes wrong. We shall formalize this below using automata as a familiar model to many readers.

Starting at a low level, change can be modelled by a finite automaton, in which the transitions are operators $\hat{O}$ as above. The changes applied after a finite series of steps can be represented as a matrix product of the form $\hat{O}_1 \cdots \hat{O}_n$.

In automaton theory, one makes the distinction between deterministic and non-deterministic automata[18]. A deterministic automaton is a 5-tuple

$$M_D = (Q, A, |q_i\rangle, Q_f, \hat{\Delta}_D), \tag{21}$$

where Q is a set of states, A is an alphabet of input instructions, q_i is an initial state, $Q_f \in Q$ a set of possible final states and $\hat{\Delta}_D : Q \times A \to Q$ is a transition function that takes the automaton from a current state to its next state, deterministically in response to a single input symbol from the alphabet. Such a string of operational symbols is the basis for a 'journal' that is intended to track the changes made. In automata which form 'sufficiently dense' graphs, the transition function's effect may also be seen as an evolution operator, driving the system through a path of states

$$\hat{\Delta}_D(I) : |q\rangle \to |q + \delta_I q\rangle \tag{22}$$

This mapping might or might not be a bijection; it might or might not possess an inverse. Although automata are considered to be a model for computation (as well as for parsing patterns or grammatical structures), they can be used to describe change at any 'black box' level of system description, as the model is entirely general.

A non-deterministic automation is almost the same as a deterministic one, except that its transition function $\hat{\Delta}_N : Q \times \{A \cup 0_+\} \to Q$, accepts one more pseudo-symbol, 0_+, which is the empty input string. Thus, a non-deterministic automaton can make transitions spontaneously, unprompted by input. In the language of [8], a deterministic automation is a *closed system* (one that is self-determined, with complete information) and a non-deterministic one is an *open system* (one that is only partially self-determined, with incomplete information and experiences 'ghostly' external influences).

Observe that automata can measure time (a clock is merely a simple deterministic automaton whose ticks are changes of state), but that such an automaton can miss events that happen between its ticks (this point will actually be key to understanding why rollback cannot be guaranteed). Thus to be able to speak of times in the following arguments, we shall assume the existence of a clock that makes transitions as fast or faster than any other transitions in the system. Such a clock exists in all human-built digital computer systems, of course.

Proposition 2 (Identification of time) *It is possible to meaure the time at which any change, deterministic or non-deterministic, occurs by means of an external deterministic automaton, called a clock.*

We shall henceforth assume that we can attach times to any change of state of the system. Non-deterministic changes are, of course, common in real systems; here are some examples:

- Interruption of a system by a key press or mouse movement.
- Deletion of all files from a computer.
- Removal of a firewall: system is infected by virus, replace firewall, system is still infected by virus.
- Checkpointing: erase state and replace with a stored image from time t_0, all history is lost between t_0 and now.

In each of these cases absolute change (overwriting) is a key theme: like zeroing out the registers in a computation. There is a one-to-one correspondence between input and output only in the deterministic case.

Proposition 3 (A computer system) *For the remainder of the discussion, we shall assume that a 'computer system' is a non-deterministic automaton, represented as a set of states, classified into categories of representation (i.e. data types) based on fields, and that these are isomorphic, through extension, to the rational numbers.*

We thus assume non-determinism of all systems, because modern, preemptive operating systems exhibit high level changes to observable data objects that cannot be traced to an alphabet of intentionally applied and documented operations. Thus there are apparent transitions that cannot be explained by any 'journal'.

7 Journals and histories of change

In this section we discuss how intended change and unintended system change can be tracked and recorded in order to maintain complete information about a system's states over time. We shall distinguish a the concept of a journal (intended change) from that of a history (which includes all changes intended and unintended). We begin by discussing relative increments.

In any solution generated by a difference equation (or transition function), the conversion of small increments or 'deltas' into an absolute state requires the specification of end-points, analogous to the limits of a contour integral in calculus along a well-defined path P:

$$q_f - q_i = P \int_{q_i}^{q_f} dq, \tag{23}$$

where q_i is the initial-state and q_f is the final state. This path corresponds to a sequence of input symbols for an automaton, corresponding – in our case – to operators to be applied. The analogue in terms of group transformations is to start from an origin state, or 'ground state' $|0\rangle$ (often called a baseline state in system operations), and to apply relative changes sequentially from this to achieve a final desired outcome.

The choice of the baseline state lies outside of the specification of the change calculus. The origin or baseline state is an ad hoc fixed point of the system, by virtue of an external specification alone. It is arbitrary, but usually plays a prominent role in system operators' model of system change. In this work, the choice of a baseline state is part of what we shall refer to as a calibration of the system, but counter to tradition we shall advocate calibration of the *end state* rather than the ad hoc *initial state*.

To model intended versus actual change, we introduce the notion of a journal, inspired by the notion of journaling in filesystems. A journal is a documentation of changes applied to a system intentionally, noting and remembering that – in real systems – this can be different from what actually takes place.

Definition 1 (Journal $\hat{J}$) *A complete, ordered sequence of all input symbols passed to an automaton α^* from an initial time t_i to a final time t_f is called the automaton's journal $\hat{J} = (\alpha^*, t_i, t_f)$. Each symbol α corresponds to a change in system state $\delta_\alpha q$. A journal has a scope that is known to the user or process that writes the journal. A journal change δJ involves adding or removing symbols in α to J, and adjusting the times. For brevity, we shall use the notation $\hat{J}(t_i, t_f)$ for the journal for a single automation starting at time t_i and ending at t_f.*

Note that, in the interests of notational simplicity, we suppress all labels relating to automata in writing $\hat{J}(t_1, t_2)$, assuming that all changes are relative to the same automaton. We use the times at which the journal began and ended, as measured by some hypothetical external clock, and the record of all known transitions is the journal itself.

Two journals $\hat{J}_1$ and $\hat{J}_2$ may be called *congruent* if they have the same number of symbols $|\hat{J}_1| = |\hat{J}_2|$ and every symbol is identically present and in the same order[19].

Lemma 1 *The final state $|q_f\rangle$ obtained by applying congruent journals of transitions $\hat{J}_1, \hat{J}_2$ to identical automata M_1, M_2 is identical, iff the initial states $|q_i\rangle$ are identical, and M_1 and M_2 are deterministic.*

This follows from the definitions of (non-)deterministic automata which allows spontaneous changes $\delta_{0+}q$. To record all changes in a non-deterministic system we need to record absolute state even when no input change is made. This brings us to:

Definition 2 (History) *A complete, ordered stack of all intermediate snapshots of a system's total state $|q_f\rangle(t)^*$ output by an automaton at all times t between an initial time t_i to a final time t_f is called the automaton's history H. A change δH involves pushing or popping the complete current state onto the stack H, and adjusting the times.*

The history H is capable of including states that were not directly affected by the journal transitions $\delta_\alpha q$. We use a stack as a convenient structure to model histories; see for instance [20] and references for a discussion of stacks. The ability to model system configuration by relative changes is affected by the following lemma:

Lemma 2 *For any automaton M, $|H_M| \geq |\hat{J}_M|$, and $|H_M| = |\hat{J}_M|$ iff M is a deterministic automaton (closed system).*

The proof follows from the form of the transition functions for automata, and the possibility of one or more occurrences of 0_+ in the input of a non-deterministic automaton. In a deterministic system each α leads to a unique labeled transition $\delta_\alpha q$, and vice versa. In the non-deterministic case, the history can contain any number of changes δ_{0_+} in addition to the α, thus the length of the history is greater than or equal to the length of the journal.

A journal is thus a sequence of *intended* changes, whereas a history is a sequence of actual changes.

Definition 3 (Roll-back operation $\hat{J}^{-1}$) *The inverse application of a string of inverse journal operations is called a roll-back operation, such that $\hat{J}^{-1}(t_3, t_2) J(t_2, t_1) = I_J$, with no intermediate symbols allowed, and $I_J \equiv \hat{J}(t,t) = (\emptyset, t, t)$. The inverse is said to exist iff every operation symbol in the journal has a unique inverse.*

For example, for relative change:

$$\hat{J}(t_q, t_{q'}) : |q\rangle \quad \mapsto \quad |q + \delta q_1 + \delta q_2 + \delta q_3\rangle \equiv |q'\rangle \tag{24}$$

and

$$\hat{J}^{-1}(t_q, t_{q'}) : |q'\rangle \quad \mapsto \quad |q' - \delta q_3 - \delta q_2 - \delta q_1\rangle \equiv |q\rangle \tag{25}$$

Lemma 3 *A roll-back journal $\hat{J}^{-1}(t_i, t_f)$, for automaton M, starting from state $|q_f\rangle$ will result in a final state $|q_i\rangle$ iff M is deterministic and J^{-1} exists.*

Proof 1 *Assume that M is non-deterministic; then the transition to state q_f is only a partial function of the journal $\hat{J}$, hence $\hat{J}^{-1}$ has more than one candidate value and thus cannot exist. If M is deterministic then the inverse exists trivially by construction, provided that each operation in the journal exists.*

Setting aside technical terminology, the reason for a failure to roll-back is clearly the loss of correspondence between journal and history caused by changes that happen outside the scope of the intended specification. This loss of correspondence can happen in a number of ways, and (crucially) it is likely to happen because today's computer systems are fundamentally non-deterministic[1].

Definition 4 (Commit and Restore operations) *A commit operation at time t is a system change $\hat{g}$ followed by a push of current history state onto a stack as consecutive operations:*

$$commit(t) : (\hat{g}, push(|q\rangle(t))) \tag{26}$$

A restore operation is a sequence of one or more operations:

$$restore(t) : pop(|q\rangle) \tag{27}$$

These operations are typical of version control schemes, for example. The importance of this construction is that previous states can be recaptured regardless of whether the operation $\hat{g}$ is invertible or not.

[1]In [8], it was pointed out that this mirrors results in information theory[21] about transmission of data over noisy channels, for which one has the fundamental theorem of channel coding due to Shannon[22] that enables the re-assertion of correspondence between a journal (transmitted data) and actual history (received data) over some time interval. However, we shall not mix metaphors by pursuing this point here.

Lemma 4 *For automaton M, n consecutive restore operations starting from t_f, are the inverse of n consecutive commit operations ending at t'_f, iff the journal of changes between $t_f > t'_f$ and t'_f is empty and M is a deterministic automaton.*

The proof, once again, follows from the absence of uncaptured changes. If $t'_f > t_f$ and the journal is empty then the only changes that can have occurred come from symbols 0_+, but these only occur for non-deterministic M. We add the following to this:

Lemma 5 *A system journal $\hat{J}$ cannot be used to restore system state for arbitrary changes $\hat{g}$.*

This result is clear from the independence of the restore operation on $\hat{g}$, and the lack of a stack of actual state in a journal $\hat{J}$.

What the foregoing discussion tells us is that there is no predictable outcome, either in a forward or a reverse direction, in an open (non-deterministic) system using relative change, and that a journal is quite useless for undoing changes that have no inverse. System configuration is analogous to making calculations in which variables change value spontaneously (as in fact they do without error correction at the hardware level). To make change computation predictable, we need to fix the outcomes rather than the sequences of operations, using 'singular change operations' for computing the final state. This was the main observation learned in the development of Cfengine[9, 10, 15].

8 Singular transitions and absolute change (overwriting)

In the foregoing cases, the initial choice of state $|q_i\rangle$ was external to the specification of the change, and was the 'origin' of a sequence of changes in a journey from start to finish. This relative ('sequential process') approach to change is deeply in-grained in management and computing culture, but it fails to bring the require predictability due to underlying system indeterminism. The problem is the reliance on the $+$ operation to navigate the state space, so our next step is to suppress it.

Now consider a class of transitions that are not usually considered in classic finite state machines. These are (non-invertible) elements $\hat{p}$ with the property that $\hat{p}|q\rangle = |q_0\rangle$, for any q. The final states are 'eigenstates' of these singular group operations: $\hat{p}|q_0\rangle = |q_0\rangle$. These effectively demote the explicit reliance on $+$ and replace it with a linear function.

Definition 5 *A singular transition function $\hat{C}_{|q_0\rangle}$ is a transition from any state $|q\rangle$ to a unique absorbing state $|q_0\rangle$. It is a many-to-one transition, and is hence non-invertible without a history.*

Such transition functions (operators) were introduced in [10] and described in [9], as an alternative to relative change to restore the predictability of outcome. These 'convergent operations' are based on fixed points or eigenstates of a graph. They harness the property of zero elements to ignore the current and historical states and to install a unique state regardless of the history or the determinism of the system. Such parameterized operators form a semi-group $\hat{C}_{|q_0\rangle}$ with the abstract property:

$$\begin{aligned} \hat{C}_{|q_0\rangle}|q\rangle &= |q_0\rangle \\ \hat{C}_{|q_0\rangle}|q_0\rangle &= |q_0\rangle. \end{aligned} \tag{28}$$

For ease of notation in the following, we drop the $|q_0\rangle$ subscript and write $\hat{C}$ for $\hat{C}_{|q_0\rangle}$

The price one pays for this restoration of predictability is an inability to reverse the change. Let us suppose that an object $\hat{C}^{-1}$ exists such that $\hat{C}^{-1}\hat{C} = I$, satisfying the latter equation. Then operating on the left, we may write using (28):

$$\begin{aligned}\hat{C}\,|q_0\rangle &= |q_0\rangle \\ \hat{C}^2\,|q_0\rangle &= \hat{C}\,|q_0\rangle \\ \hat{C}^{-1}\hat{C}\,|q_0\rangle &= \hat{C}^{-1}\,|q_0\rangle \end{aligned} \tag{29}$$

Thus, at $|q_0\rangle$ we have idempotence and a constraint:

$$\hat{C}\,|q_0\rangle = \hat{C}^{-1}\,|q_0\rangle \tag{30}$$

$$\hat{C}\,|q_0\rangle = \hat{C}^2\,|q_0\rangle. \tag{31}$$

The latter result (31) is independent of the existence of an inverse. For a ring, this condition is equivalent to the 'restricted inverse law' used in [12, 13], and it tells us that the inverse would have to be either 0 or 1.

Lemma 6 *The operators $\hat{C}_{|q_0\rangle}$ are idempotent and converge on a fixed point final state $|q_0\rangle$.*

This follows immediately from eqn (31). The value of these operations is that they can be iterated endlessly, with predictable outcome, in the manner of a highly compressed system error-correction process.

Example 1: One can view the state $|q\rangle$ as embodied in the operator $\hat{C}_{|q\rangle}$ and thus view $\hat{C}_0$, $|q\rangle$, and $|q_0\rangle$ as elements of the same semigroup. Then we may write:

$$\hat{C}_0|q\rangle = |q_0\rangle \tag{32}$$

$$\hat{C}_0|q_0\rangle = |q_0\rangle. \tag{33}$$

Assuming an additive inverse for each element (in the statespace), and subtracting these equations for arbitrary q leads to the conclusion that $\hat{C}_0 = |q_0\rangle = |0\rangle$, thus there is only a single object with this ability to take an arbitrary initial state and render a predictable outcome. Note that, in this representation, $\hat{C}$ and $|q\rangle$ belong to the same semigroup of scalars. So, choosing $|x\rangle = \hat{C}_0 = |q_0\rangle$, we service

$$(x^{-1}\cdot x)x = x \tag{34}$$

using (33). This is the restricted inverse law for fields[12, 13].

The zero plays a fundamental role as an eraser. The uniqueness of zero is not an impediment to using the zero element as a 'policy operator' which sets an intended state, as we are free to construct a homomorphism $h(|q\rangle)$ which *calibrates* or shifts the absolute location of the solution: e.g.

$$\begin{aligned}\hat{C}_0 h(|q\rangle) &= h(|q_0\rangle) \\ \hat{C}_0 h(|q_0\rangle) &= h(|q_0\rangle)\end{aligned} \tag{35}$$

where q_0^* represents a new calibration point, and $h(|q\rangle) = |q - q_0^*\rangle$, yielding the solution $C_0 = 0, q_0 = q_0^*$.

Example 2: Consider the tuple form used earlier, and let

$$\begin{aligned} C_0 &\mapsto \begin{pmatrix} 0 & q_0 \\ 0 & 1 \end{pmatrix} \\ |q_0\rangle &\mapsto \begin{pmatrix} q_0 \\ 1 \end{pmatrix}\end{aligned} \tag{36}$$

Subtracting the equations using this representation for C_0 and q leads to a result that is identically true, hence we are free to choose the value of q_0 as a matter of policy. However, one observes that C_0 does not possess a defined inverse according to the normal rules of fields.

Lemma 7 *Any inverse representation of a linear operator, taking values in a field F and satisfying $C^{-1}C = I$ and (28) must involve division by zero.*

Proof 2 *Let C be a linear function of an underlying field, then C has the form $C(c)||q\rangle \to |mq+c\rangle$, and thus the inverse is $C^{-1}(c)|q'\rangle \to |(q'-c)/m\rangle$. In order to satisfy (28) for all q, we must have $m = 0$.*

This shows that the 'rollback' of an absolute state change is directly analogous to a division by zero in a ring computation. The reason is clear: both are attempts to reverse a transition after dumping all history of initial state. We must therefore conclude that either such operations are irreversible, or that an inverse must be constructed for completeness, subject to other limitations.

9 Computing and reversing absolute states

Fields have only one element with singular properties: the zero element. It plays two distinct roles: as an identity element for the $+$ operation, and as a fixed point in scaling under $\cdot$. As a fixed point, zero annihilates state, since $0q = 0$ for any q. The zero element thus ignores and deletes any history that led us to the state q.

It is useful to think of the C operators in the above as a kind of zero-element: they annihilate state in a similar way. The utility of the convergent operations for bringing about absolute change is such that it is useful to embed them in the formalism of a general field structure for computation. One motivation for this is the recent work by Bergstra and Tucker of totalization of fields, and 'Meadows', in which they replace the partial function (excluding 0 for division) at the heart of field computation with one that is total, up to constraints. We find their construction intriguing and highly relevant to the matter of reversibility of state. As in Bergstra and Tucker, we reason by first defining the algebraic signatures of structures.

We construct an image $Alg(\Sigma_\Phi, E_\Phi)$ of a field F, with initial algebra $Alg(\Sigma_F, E_F)$ (as yet a regular field), by introducing a map in three piecewise partial representations $\Phi_F = \{S_C, S_\Delta, S_\mu\}$, where S_C is a set of linear partial-functions $C(F)$, and S_μ is a set of linear functions $\mu(F \neq 0)$, etc, to be explained below. The signature then contains only product explicitly, as addition is concealed as multiplication as described in section 5.

$$
\begin{array}{rcl}
\textbf{signature } \Sigma_\Phi & & \\
\textbf{sorts} & : & S_C, S_\Delta, S_\mu \\
\textbf{constants} & & \\
I_\Delta & : & \to S_\Delta \\
I_\mu & : & \to S_\mu \\
\textbf{operations} & : & \\
\cdot & : & S_\Delta \times S_\Delta \to S_\Delta \\
\cdot & : & S_\mu \times S_\mu \to S_\mu \\
\cdot & : & S_C \times S_C \to S_C. \\
\textbf{end} & &
\end{array}
\tag{37}
$$

We define E_Φ as the image of E_F for the field by,

$$
\begin{array}{lrcl}
 & \textbf{equations } \Phi_F & & \\
(\forall X \in S_\Delta) & X^{-1}X & = & I_\Delta, \\
(\forall Y \in S_\mu) & Y^{-1}Y & = & I_\mu
\end{array}
$$

$$\begin{array}{rrcl}
(\forall X_1, X_2 \in S_\Delta) & X_1X_2 & = & X_2X_1 \\
(\forall X_1, X_2, X_3 \in S_\Delta) & X_1(X_2X_3) & = & (X_1X_2)X_3 \\
(\forall Y_1, Y_2, Y_3 \in S_\mu) & Y_1(Y_2Y_3) & = & (Y_1Y_2)Y_3 \\
(\forall Z_1, Z_2 \in S_C) & Z_1Z_2 & = & Z_1 \\
(\forall Z_1, Z_2, Z_3 \in S_C) & Z_1(Z_2Z_3) & = & (Z_1Z_2)Z_3. \\
 & \textbf{end} & &
\end{array} \tag{38}$$

An example of the linear functions in the matrix representation is given by:

$$\Delta(x) = \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix} \tag{39}$$

$$\mu(y) = \begin{pmatrix} y & 0 \\ 0 & 1 \end{pmatrix} \tag{40}$$

$$C(z) = \begin{pmatrix} 0 & z \\ 0 & 1 \end{pmatrix} \tag{41}$$

where $\Delta^{-1}(x) = \Delta(-x)$, $\mu^{-1}(y) = \mu(1/y)$, $y \neq 0$, and we may note that $I_\Delta = \Delta(0)$, $I_\mu = \mu(1)$, and $\mu(0) \notin \mu$ in matrix terms. The function C, in any representation, gives us a way of representing absolute, not relative, changes of state, and contains the equivalent of 'zero'. This is an important ability in maintaining order in a system, and it is the basis on which Cfengine[10] operates on millions of computers around the world today. Each operation is a function of a field, in which the zero element is mapped to a desired state. The set of all possible parameterized $C(F)$ must therefore span a field, and yet it contains no (multiplicative) inverses at all. This is the interesting paradox which plays into the work of Bergstra and Tucker. The restriction $x \neq 0 \in F$ is prominent.

It is not our intention to reiterate the arguments for totalizing fields, presented by Bergstra and Tucker[13]. As they point out, there is a number of ways to restore 'faith' in the connection between state and history of change after a zero operation, using proof systems, axioms and algebraic properties. Each brings a different kind of merit. As they remark, the issue is not so much about going backwards (reversal) as about going forwards in a way that is unaffected by an ill-defined attempt at reversal.

One remedy relies on proving the outcome of a change was not affected by the result of 0^{-1}, i.e. the final state is independent of the path taken to evaluate it. Another involves changing the definitions of computation (change) to disallow unsafe operations. Finally one might simply give up on certain requirements so that the outcome satisfies a well defined set of equations (policies) in order to prove that the result is well-defined. Here, we observe by analogy that one may:

- Introduce a stack of history-snapshots to some maximum depth[20]. (This is difficult to do in arithmetic but it is plausible for some system changes.)

  Modern filesystems have the capability to take snapshots of disk media. But this does not cover the dynamic, runtime state of a system, so this is only a partial snapshot. Virtual machines can also be snapshotted for restoration with certain limitations. In practice, snapshotting a system in isolation is a meaningless remedy because the remainder of the environment around it has already moved on, and therefore one risks the system becoming unsynchronised with its environment, e.g. in relation to protocol communications and stateful channels.

- Totalize the data type, using the notion of a totalized field, e.g. set $0^{-1} = 0$, or equivalently, $C^{-1} = C$.

  In systems, this corresponds to defining a policy for the proper state of the system, e.g. as with Cfengine. Defining a policy is analogous to defining the 'zero' of the entire system.

Undoing a zero requires itself a policy, which is just another zero (either the same of transformed by recalibration – see below).

- Perform a naive reversal and then apply some policy equational specification to clean up the result.

  In system terms, this corresponds to restoring a previous snapshot from memory and then testing the system to see if it complies with a number of constraints. These might include answering the question: are all my communications with external agents synchronized and in a consistent state?

- Abandon the attempt to introduce reversals altogether ("rollback does not exist").

  In this final, more fatalistic, approach one does not attempt to perform any inverse operations on discovering a problem, but merely recalibrates the system to a new desired state, either by relative or absolute change.

Given that relative change is fragile to incomplete specifications of a system, the above suggests that greater certainty can be achieved by adopting an absolute approach to configuration: i.e. by embracing zero.

10 Calibration of absolute states

Let us complete the formalization of the operators for absolute change. We build linear functions on top of the (totalized) field and end with a vector space. We no longer care about Δ and μ, but want to embrace the properties of the zero operators to bring predictability in a non-deterministic environment. We start with a signature for the convergent operators based on a different use of commutative rings as a parameterization of the zeroed outcome, and end with non-commutative, non-invertible representations. Let F signify a field, with the usual field axioms.

We use a Σ-algebra $\Sigma_C = \{|q\rangle \mid I,\, 0, \oplus,\, \circ,\, C\}$, and this is understood to extend the field algebra $Alg(\Sigma_F, E_F)$. Thus, for any index set labels α, β, labeling the underlying field q, we have signature:

$$
\begin{aligned}
\textbf{signature } \Sigma_C & \\
\textbf{sorts} &\;:\; F, F' \\
\textbf{constants} & \\
|q\rangle &\;:\; \to F' \\
\textbf{operations} & \qquad\qquad\qquad\qquad (42)\\
C : &\quad F \times F' \to F' \\
\oplus : &\quad C(F \times F) \to C(F) \\
\circ : &\quad C(F) \times C(F) \to C(F) \\
\textbf{end} & \qquad\qquad\qquad\qquad (43)
\end{aligned}
$$

And equations:

$$\textbf{equations } E_C :$$

$$C_\alpha |q\rangle = C(q_\alpha)|q\rangle = |q_\alpha\rangle \tag{44}$$

$$C_\alpha \circ C_\beta = C_\alpha \tag{45}$$

$$(C_\alpha \circ C_\beta) \circ C_\gamma = C_\alpha \circ (C_\beta \circ C_\gamma) \tag{46}$$

$$\begin{aligned} C_\alpha \oplus C_\beta &= C_{\alpha+\beta} && (47)\\ (C_\alpha \oplus C_\beta) \oplus C_\gamma &= C_\alpha \oplus (C_\beta \oplus C_\gamma) && (48)\\ &\textbf{end} && (49) \end{aligned}$$

Naturally, these are true for all α, β, γ, and we are working with $Alg(\Sigma_F \cup \Sigma_C, E_F \cup E_C)$. The possibly opque formalism belies a simple structure. Every state $|q\rangle$ is fully specified by a field value $q \in F$. Similarly, every convergent operator $C(q)$ is fully specified by a field value $q \in F$, and results in a new value $q \in F$, which obeys the zero property (44). Thus the C is a transformer which takes any input state and outputs a specific state given by its label (but importantly, only one at a time). This has the 'zero' property of ejecting initial state and replacing it wholesale with particular one. Clearly, the operators must be idempotent from (45).

The representation in (36) is useful to see how a tuple-representation quickly captures this algebra. We say that the repeated operation of an operator $C(q_0)$ 'converges', as it always returns the system state to its fixed point $|q_0\rangle$. This algebra describes the behaviour of a single 'convergent operator' or 'promise' in Cfengine[9, 10]. We cannot define C^{-1} because the symbol 0^{-1} is not defined in the underlying field F, but we may totalize the field[13] with corresponding merits and conditions to assign a meaning to a reversal or 'roll-back'.

11 Re-calibration - change of policy

There is only a single fixed point for each operator $C(q_0)$. What happens when we want to change the outcome of a 'promised state', i.e. change the value of q_0? The homomorphism h on states, in eqn. (35) allowed us to calibrate a single singular outcome to any field value by shifting the zero, but this is less useful than modifying the operators themselves to bring about the desired result. This transformation then has the simple interpretation as an operator the re-calibrates the system baseline.

Lemma 8 *Each operator has only one singularity, i.e. let F be a field, totalized or not, and let 0_1 and 0_2 be zero elements for $\cdot$, then $0_1 = 0_2$.*

The proof follows by substitution of the field axioms: $0_1 x = 0_1$, $0_2 x = 0_2$, setting $x = 0_2$ in the former, implies $0_1 = 0_2$. Hence the zero element is unique in a field.

This means that we cannot have more than one policy fixed point per field. In configuration terms, one cannot have more than one policy for a data item, so any path of changes parameterized by chaining q_0 can be uniquely characterised.

This leaves only the possibility of shifting the fixed point by re-calibration, or change of policy. This is no longer a journal of deltas, but a kind of 'teleportation' or 'large transformation' in the group theoretic sense. Given this, and the utility of formulating policy changes as applied operations, it is useful for the purpose of making contact with reference [8] to reformulate the values in terms of *vector spaces*. The specification of a vector space is somewhat similar to that of a ring or field except that it is not automorphic.

Let F be a field (totalized or not) and S be a set. A vector space of F is a triple $(S, +, \cdot)$, with the equations:

$$\begin{aligned} &\textbf{signature} \\ &\quad \textbf{sorts} : F, S \\ &\textbf{constants} \\ &\quad 0_S : \to S \\ &\quad 1_F : \to F \end{aligned}$$

$$\begin{array}{rcl} & \textbf{operations} & \\ + & : & S \times S \rightarrow S, \\ - & : & S \times S \rightarrow S, \\ \cdot & : & F \times S \rightarrow S, \\ & \textbf{end} & \end{array} \tag{50, 51}$$

with properties:

$$\begin{array}{rrcl} & & \textbf{equations} & \\ (\forall x \in S) & x + 0_S & = & 0_S + x = x, \\ (\forall x \in S, -x \in S) & x + (-x) & = & (-x) + x = 0_S, \\ (\forall x, y \in S) & x + y & = & y + x, \\ (\forall x, y, z \in S) & (x + y) + z & = & x + (y + z), \\ (\forall \alpha, \beta \in F, z \in S) & (\alpha\beta)z & = & \alpha(\beta z), \\ (\forall x \in S,) & 1_F x & = & x 1_F = x, \\ (\forall \alpha, \beta \in F, x \in S) & (\alpha + \beta)x & = & \alpha x + \beta x, \\ (\forall \alpha \in F, x, y \in S) & \alpha(x + y) & = & \alpha x + \alpha y, \\ & \textbf{end} & & \end{array} \tag{52, 53}$$

The usefulness of this map is that it involves an external 'promise' or 'policy' field F from which we may construct the set of C_α, not merely an automorphic image of a single set. Thus we can separate policy from changes with convergent, fixed-point zero-operators $0_A \in F_A$, all acting on a single set of states $q \in S$. We thus arrive, by a different route, at the formulation as a vector space in ref. [8].

We note finally that a change of calibration cannot be a commutative ring.

Lemma 9 *Let C_A and C_B be zeros of F_A and F_B. Then C_A and C_B cannot commute unless $A = B$.*

Proof 3 *The proof is similar to the earlier proof of uniqueness of zero in a ring. We have $C_A q = C_A$, and $C_B q = C_B$ for all q. Substituting $q = C_B$ in the former, we have*

$$\begin{aligned} C_A(C_B)q &= C_A C_B = C_A \\ C_B(C_A)q &= C_B C_A = C_B \end{aligned} \tag{54}$$

Thus the commutator

$$[C_A, C_B] = C_A C_B - C_B C_A = C_A - C_B \neq 0 \quad (A \neq B) \tag{55}$$

This proof does not depend on the representation of F_A and F_B, thus it applies equally to higher dimensional tuple formulations also.

12 Predicting outcome with roll-back-safe change

The problems of indeterminism cannot be addressed without absolute change operations, but these do nothing to repair the problem of unsafe reversals. The C operations allow us to basically forget about indeterminism, but not irreversibility. We therefore need to find an approach analogous to that of [13] during non-commutative strings of system re-calibrations. We have one advantage here: a lack of commutativity. This is in fact a strength as it makes the need for reversal practically irrelevant. In our view, there is then only one natural choice for C^{-1} or $\hat{J}^{-1}$ and that is to apply or re-apply the current policy $C(t)$: it is absolute, idempotent and it overrides any previous 'mistakes'.

We have also one disadvantage compared to [13] and that is that time is relevant: we cannot undo the potential consequences of being in a bad state unless we manage to totality of state within the system. For real computers, that might be almost the entire Internet (e.g. during the spread of viruses).

Other weaker arguments can be made for resetting state to a baseline, e.g. (i) Use an arbitrarily chosen baseline state $|q_{\rm initial}\rangle$ or $|t_0\rangle$ so that an arbitrary journal of convergent changes $\hat{J}_0$

$$\begin{aligned} |t_{\rm final}\rangle &= \hat{J}_0 |t_{\rm initial}\rangle \\ &\equiv \ldots \hat{O}_2 \hat{O}_1 |t_{\rm initial}\rangle \end{aligned} \tag{56}$$

has an inverse such that

$$J_0^{-1}|t_{\rm final}\rangle = |t_{\rm initial}\rangle. \tag{57}$$

Assuming the existence of an operator $\hat{O}_{\rm initial}$ such that $\hat{O}_{\rm initial}|q\rangle = |t_{\rm initial}$, then clearly

$$J_0^{-1} = \hat{O}_{\rm initial}. \tag{58}$$

These two choices are both forward-moving absolute changes since they both involve an arbitrary decision and they both move forward in time. However the latter is less natural, since it affects to return to a time in the past which might have nothing directly to do with where one needs to be in the present. Our study was motivated by predictability. The principal advantage of these remedies lies in knowledge of the outcome, in the absence of a complete specification.

13 Multi-dimensional operators

In the discussion above, we have restricted ourselves to the maintenance of a single scalar system-value. The issue of dependencies amongst system changes enters quickly as the complexity of layered models of a system grows. It was shown in [9] that one can develop a spanning set of orthogonal operations that covers the vector space like a coordinate system, simply by embedding in a geometrical tuple-fashion. In the simplest expression, one sees this by extending the matrix representation to higher dimensions.

One way to do this is to consider a system as a controlled by a vector of its individual configuration parameters X_i, where each parameter is embedded into the field of rationals and encoded in the obvious way:

$$|X\rangle = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \\ 1 \end{pmatrix}. \tag{59}$$

As large as current systems may be, they remain finite and can be modelled by finite vectors. We define relative operators for individual parameters in a state as

$$\Delta_i(q) = \begin{pmatrix} 1 & 0 & \cdots & & & 0 \\ 0 & 1 & & & & \\ \vdots & & \ddots & & & \\ & & & 1 & & q \\ & & & & \ddots & \\ 0 & & & & & 1 \end{pmatrix} \tag{60}$$

(where q appears in the $(n+1)$st column of the ith row). Likewise, absolute operators are defined as

$$C_i(q) = \begin{pmatrix} 1 & 0 & \cdots & & & 0 \\ 0 & 1 & & & & \\ \vdots & & \ddots & & & \\ & & & 0 & & q \\ & & & & \ddots & \\ 0 & & & & & 1 \end{pmatrix} \tag{61}$$

(where q appears again in the $(n+1)$st column of the ith row), and multiplicative operators as

$$\mu_i(q) = \begin{pmatrix} 1 & 0 & \cdots & & & 0 \\ 0 & 1 & & & & \\ \vdots & & \ddots & & & \\ & & & q & & \\ & & & & \ddots & \\ 0 & & & & & 1 \end{pmatrix} \tag{62}$$

(Where q appears in the ith row and column). Thus we propose to model configuration changes in a system via a set of matrices with rational entries.

This completes the construction of the Cfengine operators. Clearly the zero inverse solution applies independently to each of these diagonal operators in this basis, but becomes rapidly more entangled in other parameterizations, where dependencies occur.

14 Concluding remarks

We have shown that neither the outcome of a journal of changes, nor an attempted reversal (undo operation) is generally meaningful in an incompletely specified i.e. open, or non-deterministic system. A deterministic outcome can only be obtained by grounding a system in an absolute way to a policy-defined state, analogous to a ‘zero’ in a field. This is a move from a relativistic view of error correction to an absolute determination of outcome.

By formulating this problem algebraically, the discussion is distanced from the sometimes emotional standpoints that bind system administrators to the notion of rollback: desperately wanting does not make it possible. The discussion about totalisation of fields is particularly useful, as it maps nicely to the flaws in this thinking. To deal with the inverse of a many-to-one map, one must invoke a *policy* or arbitrary selection.

In computation, division by zero is ambiguous because it seeks to ‘reverse’ a number of relative transformations with an absolute change, while the latter has no definitive remedy within the scope of an expression itself. In system management, the rollback is ambiguous due to the incomplete information about transitions arrising from parallel processes, whose history is not available within the scope of the system itself. In both cases, external information is required to compensate for the incompleteness of the information. The generic remedies proposed for the abstract, algebraic totalisation of fields have clear analogues in the practical world of systems, and bring rationality to the argument.

Even keeping a history of every transition made does not solve all the issues of rollback: reversing states locally does not make time travel possible. Neither the ‘restoration of state by roll-back’ nor ‘division safe calculation’, à la Bergstra and Tucker, can take us backwards in time to undo an absolute mistake throughout the entire scope of the world: errors propagate. A local reversal might merely break the synchronisation of the local system with the remainder of the environment, e.g. trying to undo a conversation between two automata by rolling back one of them.

The arguments both here and in the work of Bergstra and Tucker thus focus on how one goes a meaningfully forwards to repair a lack of synchronicity, given that a poorly formulated reversal of a singular change was attempted. A classic answer is 'well, don't do that' – but we know that someone will always attempt to perform ill-defined operations and thus our story has considerable importance.

The remedies proposed here mirror ways in which the problem of singularities is handled in other areas of mathematics, e.g. in complex analysis one has analytical continuation[23], in which a path or history through the states can be defined such that the final result avoids touching the singular cases; similarly in algebraic topology, the uniqueness of the result can then depend on the path and cohomology. The totalization remedies described by Bergstra and Tucker thus underline an approach to a wider range of problems of incomplete information. The ultimate conclusion of this work is that 'rollback' cannot be achieved in any well-defined sense without full system closure, which is not generally possible. A choice about how to go forward is the only deterministic remedy.

For the future, there are plenty of topics we have not touched upon here.

Problem 1 *We have not taken into account fields in which external min-max boundary values are imposed on S. Then we would have further fixed points in the total history of a system to contend with:*

$$\{\min_S, 0_A, 0_B, \ldots \max_S\} \tag{63}$$

Each of these might be a reasonable candidate for 're-grounding' the system in an undo/reset – ours has been an orthogonal decomposition of the problem. What conditions might be imposed when 0_A falls outside the range $[\min_S, \max_S]$.

Problem 2 *We have not taken into account operators that depend on one another in non-orthogonal fashion[24]. Dependencies between operators add potentially severe complications to this account.*

There is a deeper issue with roll-back in partial systems. If a system is in contact with another system, e.g. receiving data, or if we have partitioned a system into loosely coupled pieces only one of which is being changed, then the other system becomes a part of the total system and we must write a hypothetical journal for the entire system in order to achieve a consistent rollback.

Problem 3 *The partial restoration can leave a system in an inconsistent state that it has never been in before and is not a state that was ever intended.*

The results in this paper are directly applicable to to hands-free automation, or 'computer immunology', as demonstrated by Cfengine. Opponents of automation have look for ways of arguing that traditional journaling approaches to system maintenance are necessary[19], preserving the role of humans in system repair. However, we argue that the role of humans is rather in deciding system policy: it is known that the computational complexity of searching for convergent operations is in PSPACE and NP complete[25, 26], thus it remains the domain of heuristic methods and system experts to find these convergent in more complex cases.

Acknowledgment: MB would like to thank John Tucker for generous advice and encouragement during the finalization of the manuscript.

References

[1] Lawrence A. Rowe and Michael R. Stonebraker. The postgres data model. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, 1987.

[2] Gerhard Weikum and Gottfried Vossen. *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Number ISBN 1558605088. Morgan Kaufmann, 2001.

[3] Kai Li, Jeffrey Naughton, and James Plank. Real-time concurrent checkpoint for parallel programs. In *Proceedings of the second ACM SIGPLAN Symposium on principles and practice of parallel programming*, pages 79–88. Association for Computing Machinery, 1990.

[4] Kai Li, Jeffrey Naughton, and James Plank. Checkpointing multicomputer applications. In *Proceedings of the tenth symposium on reliable distributed systems*, pages 2–11. IEEE Computer Society Press, 1991.

[5] James S. Plank, Youngbae Kim, and Jack J. Dongarra. Fault-tolerant matrix operations for networks of workstations using diskless checkpointing. *Journal of Parallel and Distributed Computing*, 43(2):12–138, 1997.

[6] José Nagib Cotrim Arabé, Adam Beguelin, Bruce Lowecamp, Eric Seligman, Mike Starkey, and Peter Stephan. Dome: parallel programming in a heterogeneous user environment. Technical Report CMU-CS-95-137, Carnegie Mellon University, 1995.

[7] J. S. Plank and M. G. Thomason. Processor allocation and checkpoint interval selection in cluster computing systems. volume 61, pages 1570–1590. Academic Press, November 2001.

[8] M. Burgess. On the theory of system administration. *Science of Computer Programming*, 49:1, 2003.

[9] M. Burgess. Configurable immunity model of evolving configuration management. *Science of Computer Programming*, 51:197, 2004.

[10] M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.

[11] A. Couch and Y. Sun. On observed reproducibility in network configuration management. *Science of Computer Programming*, 53:215–253, 2004.

[12] J.A. Bergstra and J.V. Tucker. The rational numbers as an abstract datatype. *Journal of the ACM*, 54:1–25, 2007.

[13] J. A. Bergstra and J. V. Tucker. Division safe calculation in totalised fields. *Theory of Computing Systems*, 43:410–424, 2008.

[14] M. Burgess. Computer immunology. *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, page 283, 1998.

[15] A. Couch and Y. Sun. On the algebraic structure of convergence. *LNCS, Proc. 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, Heidelberg, Germany*, pages 28–40, 2003.

[16] J. A. Bergstra, Y. Hirshfield, and J. V. Tucker. Meadows and the equational specification of division. *Theoretical Computer Science*, 410:1261–1271, 2009.

[17] M. Burgess. *Classical Covariant Fields*. Cambridge University Press, Cambridge, 2002.

[18] H. Lewis and C. Papadimitriou. *Elements of the Theory of Computation, Second edition*. Prentice Hall, New York, 1997.

[19] S. Traugott. Why order matters: Turing equivalence in automated systems administration. *Proceedings of the Sixteenth Systems Administration Conference (LISA XVI) (USENIX Association: Berkeley, CA)*, page 99, 2002.

[20] J. A. Bergstra and J. V. Tucker. The data type variety of stack algebras. *Annals of Pure and Applied Logic*, 73(1):11–36, 1995.

[21] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. (J.Wiley & Sons., New York), 1991.

[22] C.E. Shannon and W. Weaver. *The mathematical theory of communication*. University of Illinois Press, Urbana, 1949.

[23] M.R. Spiegel. *Theory and problems of complex variables*. Schaum Publishing Co., New York, 1964.

[24] A. Couch and N. Daniels. The maelstrom: Network service debugging via ”ineffective procedures”. *Proceedings of the Fifteenth Systems Administration Conference (LISA XV) (USENIX Association: Berkeley, CA)*, page 63, 2001.

[25] M .Burgess and L. Kristiansen. *Handbook of Network and System Administration*, chapter On the Complexity of Change and Configuration Management. Elsevier, 2007.

[26] Yizhan Sun and Alva Couch. *Handbook of Network and System Administration*, chapter Complexity of System Configuration Management. Elsevier, 2007.