SOFTWARE SAFETY DEMONSTRATION AND INDEMNIFICATION

ODELL HEGNA
hegnaoc2@gmail.com

ABSTRACT. In cyber-physical systems software may influence safety-significant operations. Statistical testing implies algorithms must be evaluated to quantify the strength of evidence that program logic performs correctly. One form of statistical testing involves sampling algorithmic behavior in a specific area of safety risk known as a hazard. When this sample is randomly drawn, it is called a safety demonstration. It provides hard evidence for indemnification, a statistic expressing a probabilistic upper bound for accident frequency. Thus indemnification expresses an assurance metric, and it is important to obtain usable results with practical sample sizes. This essay uses automata theory as an equivalent substitute for programming languages. It explains why the probabilistic upper bound arises mathematically from automata theory and other premises.

1. PROLOGUE

1.1. Copyright. This document may be freely copied or modified in accordance with the Creative Commons Attribution license.

1.2. Apologies. This essay is not rendered to academic standards of quality; it benefits from no formal literature search and was written in isolation. The experienced reader may find terms in nonstandard context. The author has strived to maintain consistency, but admits deficiency in standardization of terminology. The author apologizes for resulting inconvenience.

The author also apologizes that the concepts discussed here are nascent. Difficult engineering must be accomplished before a mature technology is available for commercialization.

The author features mathematics centrally, presuming undergraduate background and providing necessary computer science. This approach risks estranging many worthy engineering readers; however, a mathematical foundation is necessary. This essay serves that need.

2. INTRODUCTION

2.1. Purpose. In systems of integrated hardware and software, the intangible nature of software raises the question of fitness in roles bearing safety risk. During its development, such software merits not only properly controlled general engineering process, but also quantitative assurance measures particular to hazards within a specific product.

The topic of this essay is assuring the interplay between safety constraints (requirements) and software control. Software is appreciated as a branching process whose permutations are intractably numerous to test exhaustively. Barring exhaustion, statistical verification remains an option. Of particular importance is how much assurance can be obtained with practical (small) sample sizes. Needed is a testing methodology along...
with an auditable assurance metric. This essay proposes a method that can be used in the presence of product risk to measure confidence that software conforms to specification. It consists of a paired reliability demonstration and a statistic called indemnification.

2.2. **Significance.** The subject matter results from applying standard mathematics to a well-known (but cloudy) problem. It is organized according to a mathematized version of the Joint Software Systems Safety Engineering Handbook[6] of the United States Department of Defense (2010). This mathematization, however, affords a deeper structural view of safety engineering which inspires unification of that document’s separately described hardware and software disciplines. Analogous to hardware, a hazard in software is a region of code involving safety constraints (requirements) necessitating some degree of verification, usually identified and monitored by safety engineers, and possessing hypothetical (threatened) frequency and severity ratings. The ensuing uniform concepts, obscure without mathematization, emerge and lead to homogeneous analysis and management of hazards.

2.3. **Systems.** In his 1962 treatise on systems engineering methodology [2] p. 60ff., Hall gives succinct definitions of the terms *system* and *environment*:

- A system is a set of objects with relationships between the objects and between their attributes.
- For a given system, the environment is the set of all objects outside the system: (1) a change in whose attributes affect the system and (2) whose attributes are changed by the behavior of the system.

2.4. **Classifications (Mechanism, Construct, Model).** In a system, the terms mechanism, construct, and model have specially differentiated meaning. Mechanisms are abstractions whose underlying structure suffices to emulate all behaviors of a given phenomenon. Constructs use the infrastructure of mechanisms to elucidate resultant behaviors. Models interpret a behavior of a mechanism in terms of alternate infrastructure. *Exempli gratia*, hardware and software are mechanisms and operational profiles are constructs, while safety risk is a model. A description of major mechanisms, constructs, and models follows.

2.4.1. **Hardware mechanism.** The dynamics of hardware components is portrayed as constrained real time trajectories over a state space. A trajectory is a mapping from time into state space. A constraint relation is an alternative expression for what is familiar as an equation or inequality of state; it is merely a substitute for an equivalent equation. It is characteristic of systems that at any time, intersecting constraints delimit apparently independent choices so that just one is valid. Interacting constraints endow hardware with capabilities. Constraints can be classified according to their engineering significance. When violated, a safety constraint brings hazard to equipment, health, life, or surroundings.

2.4.2. **Software mechanism.** Commercialization of software safety demonstration/indemnification technology demands implementing tools for analyzing full-fledged software programming languages. However, theoretical investigations of reactive systems[3] can be accomplished using a simpler substitute: the automaton. Automata are purely mechanicistic structures positioned in the machine/language spectrum somewhere between the Turing machine and the Gurevich abstract state language (ASM). Beside its adequacy for examining theory, automata avoid selection of a programming language, which would unnecessarily particularize concepts intended to be general.

Automata perform work in discrete units called steps. A sequence of steps is further known as a walk. This essay presents the actuated automaton, a variant form whose work is deterministic conditional sequencing and application of instructions. The natures of instructions are represented by mathematical morphisms, collectively known as functionalities. The order of these functionalities is governed by the actuated automaton through its state. Iteration of an actuated automaton emulates an operating program. Because its role

---

3 A reactive system responds to external stimuli.
is so central, and because the author was unable to find references, material on the actuated automaton is
mathematized lopsidedly with respect to other topics.

2.4.3. **Reactive mechanism.** Reactive systems characteristically need some means to transfer external stim-
uli. The reactive mechanism contains structures enabling the hardware and software mechanisms to inter-
operate cohesively. The nature of time differs between the two; time is a continuum in hardware while it is
discrete in software. The clock synchronization permits integration by specifying an ordered cross-reference
between discrete and real time.

Remaining is need for inter-mechanism communication. Two forms exist:

- Sensors convey information about the hardware environment to the software mechanism. Using the
clock synchronization, a real-time trajectory is sampled into a sequence of events.
- Transducers map a point of the software state into a trajectory in hardware. This trajectory is called
a control.

2.4.4. **Cone construct.** The actuated automaton has a generalized inverse called the converse. Through reit-
eration, the converse constructs a partially ordered set (poset) of effects and potential causes. This poset is
not linear because an effect may have more than one preceding cause.

A cone is the result of decomposing the poset into constituent chains called reverse walks. Viewed as forward
walks (reversing the reverse walks), these chains are ordinary sequences of causes and consequent effects.
The collection of forward chains converges to a point known as the crux, while the cone diverges from the
same point. One subcomponent of a cone is its edge, which is the collection of steps radially opposite the

2.4.5. **Operational profile construct.** As previously mentioned, automata accomplish work in units called
steps. An operational profile is a measure of a step’s excitation probability relative to a reference set of steps.

The “reference set of steps” itself historically represented a software usage pattern, so it sought to resemble
the natural mix of functionalities in deployed software. This idea is abstracted to a potentially purposeless
reference set, but the software usage pattern remains important. From the usage pattern, along with the
automaton’s static logic, arises the very notion of probability.

An operational profile may be applied to the edge of a cone.

2.4.6. **Safety risk model.** Accidents occur haphazardly with varying frequency and severity. In the context
of software, risk expresses the impact of algorithmic design errors. Since its true extent is unknown, software
safety risk is expressed as a statistical hypothesis. The compound Poisson process is a model simulating
discrete event-based losses that accumulate with passing time. It offers the advantage of independent param-
eterization of the loss’ intensity (frequency) and severity. Indemnification is statistical assurance of software
safety.

2.5. **Principle of emergence.** Emergence [9] is a broad principle of physics describing a process whereby
larger entities possessing a property arise through interactions between simpler entities that themselves do not
exhibit the property. Particularized to software testing, the “principle of (weak) emergence” is that erroneous
software can do no actual harm until certain of its values emerge from the realm of digital logic into a physical
subsystem. This principle inquires both into mechanics of transduction, and how transducible values come
into being. The automaton of the software mechanism answers the latter question. If software hazard is
to be evaluated starting at points of transduction and proceeding backwards through internal logic, then the
automaton must support reverse inference – meaning reversed in computational order, from final conclusion
to possible premise (see [2.4.4]).
2.6. **CHOICE FORK.** Sections 3 – 8 detail certain relationships between automata theory and testing. From a mathematical standpoint the material is necessary, but there are readers for whom these sections would duplicate existing knowledge. They are invited to skip forward to section 9 on operational profiles.

Sections 3 – 8 describe software using three structures: the process, the procedure, and the automaton. Rudiments underlying these structures consist of ensembles and Cartesian products. From these beginnings follow choice spaces, walks, converse automata, reverse walks, and cones.

Readers desiring further introduction to these fundamental structures may access Appendix A which reviews groundwork and notation used here. Its highlights include that an ensemble is a mapping from a finite non-empty set (the indices) into another set of non-empty sets (the values). Ensembles are denoted by uppercase Greek letters such as \( \Psi \). The general Cartesian product of an ensemble, called a choice space, is denoted \( \prod \Psi \). If two ensembles have disjoint domains, their union \( \Psi \cup \Upsilon \) is written as the dyadic product \( \Psi \Upsilon \).

### 3. Process

Chains of stimulus and response characterize reactive discrete systems. In this chain, successive links are not independent: the response effected in one link feeds forward into the stimulus of the following link. For instance, in a system of cog-wheels and escapements, gear train movement accomplished in one stage of operation becomes input to the next. A formalism called a process captures this notion of sequential inheritance. We assemble processes from a simple unit called the frame, which is two-part structure consisting of operation becomes input to the next. A formalism called a process captures this notion of sequential inheritance. A process is a sequence of frames such that the starting condition of each frame subsumes the ending condition of its predecessor frame. Interpreted in systems language, a frame’s starting condition is a stimulus and its ending condition is a response. Current response re-appears as part of future stimulus.

**Definition 3.1.** The pair of ensembles \( \langle \Psi, \Phi \rangle \) is a basis if \( \Phi \subseteq \Psi \).

It is necessary to represent states (variables) which are used but not set – so-called “volatile” variables. For example, such variables can hold the transient values of sensors. The remainder \( \Psi \setminus \Phi \) is the generating ensemble of volatile variables (see terminology following definition A.10).

**Definition 3.2.** The frame space \( F \) of basis \( \langle \Psi, \Phi \rangle \) is the set \( \prod \Psi \times \prod \Phi \). A member \( f \in F \) is a frame.

**Terminology.** Let \( f = (\psi, \phi) \in \prod \Psi \times \prod \Phi \) be a frame. The choice \( \psi \in \prod \Psi \) is the frame’s starting condition (abscissa) and \( \phi \in \prod \Phi \) is the frame’s ending condition (ordinate).

Two frames may be related such that the ending condition of one frame is embedded within the next frame’s starting condition. This stipulation is conveniently expressed as a mapping restriction:

**Definition 3.3.** Let \( \langle \Psi, \Phi \rangle \) be a basis with frames \( f = (\psi, \phi), f' = (\psi', \phi') \in \prod \Psi \times \prod \Phi \). Frame \( f \) conjoins frame \( f' \) if \( \psi' \mid \text{dom } \Phi = \phi \).

**Notation.** A sequence in a set \( S \) is some mapping \( \sigma : \mathbb{N} \rightarrow S \) – that is, \( \sigma \in S^{\mathbb{N}} \). The anonymous sequence convention allows reference to a sequence using the compound symbol \( \{s_n\} \), understanding \( s \in S \). Formally, the symbol \( s_i \) denotes that term \( (i, s_i) \in \{s_n\} \). The convention is clumsy expressing functional notation; for instance \( s_i = \{s_n\}(i) \) means \( i \rightarrow s_i \).

**Definition 3.4.** Let \( \langle \Psi, \Phi \rangle \) be a basis with sequence of frames \( \{f_n\} : \mathbb{N} \rightarrow \prod \Psi \times \prod \Phi \). The sequence is successively conjoin if \( f_i \) conjoins \( f_{i+1} \) for each \( i \geq 1 \).

**Definition 3.5.** With \( \langle \Psi, \Phi \rangle \) a basis, a process is a successively conjoin sequence of frames \( \mathbb{N} \rightarrow \prod \Psi \times \prod \Phi \).

**Definition 3.6.** Let \( \langle \Psi, \Phi \rangle \) be a basis with frame space \( F = \prod \Psi \times \prod \Phi \). Define the abscissa projection \( \text{absc} : F \rightarrow \prod \Psi \) by \( (\psi, \phi) \mapsto absc \psi \). Define the ordinate projection \( \text{ord} : F \rightarrow \prod \Phi \) by \( (\psi, \phi) \mapsto ord \phi \).
**Definition 3.7.** Let \( \langle \Psi, \Phi \rangle \) be a basis with persistent-volatile partition \( \Psi = \Phi \Xi \) (see appendix \textsection A.4). Suppose \( f \) is a frame in \( \prod \Psi \times \prod \Phi \). The **reactive** state of frame \( f \) is \( \psi = \phi \xi = \text{absc} f \). The **event** or **volatile** excitation state of frame \( f \) is \( \xi = (\text{absc} f) | \text{dom} \Xi \). Similarly, the **persistent** state of frame \( f \) is \( \phi = (\text{absc} f) | \text{dom} \Phi \).

**Terminology.** Process concepts interpret into systems language. The reactive space \( \prod \Psi \) contains the system stimulus. Sequential conjointness allows circumstantial interpretation of the choice space. To place \( \prod \Phi \) in context of the frame’s reactive state, the Cartesian product \( \prod \Phi = \prod (\Psi \mid \text{dom} \Phi) = (\prod \Psi) \mid \text{dom} \Phi \) [by theorem A.23] is the persistent state space. Using this nomenclature, sequential conjointness is summarized that each frame’s response becomes the next frame’s persistent state, symbolically \( \text{ord} f_i = \text{absc} f_{i+1} \mid \text{dom} \Phi \).

**4. Procedure**

The procedure is useful to portray a process frame as a transformation from the stimulus space to the response space. To distinguish such transformations from other mappings, we use the special term “functionality” and stipulate that the collection of functionalities is a finite set called a catalog. The term “catalog” will later be applied to resource sets identified with an automaton.

**4.1. Functionality.** The functionality generalizes the frame. If \( f_i = (\psi_i, \phi_i) \) is the \( i^{th} \) process frame, this concept permits writing \( \phi_i = f_i(\psi_i) \), where \( f_i \) is some functionality belonging to catalog \( \mathcal{F} \).

**Definition 4.1.** A **functionality** is a mapping whose domain and codomain are choice spaces (definition A.10), with the codomain a subspace (definition A.22) of the domain.

**Lemma 4.2.** Let \( \langle \Psi, \Phi \rangle \) be a basis. Any mapping \( f : \prod \Psi \to \prod \Phi \) is a functionality.

**Proof.** As a basis, definition A.11 establishes that \( \Psi \) and \( \Phi \) are ensembles with \( \Phi \subseteq \Psi \). Since \( \Psi \) and \( \Phi \) are ensembles, definition A.10 asserts that \( \prod \Psi \) and \( \prod \Phi \) are choice spaces. Theorem A.27 provides that \( \prod \Phi \) is a subspace of \( \prod \Psi \) because \( \Phi \subseteq \Psi \). By virtue of \( f \in \prod \Phi^{\prod \Psi} \), then \( f : \prod \Psi \to \prod \Phi \) is a mapping from one choice space to another, which is a subspace of the first. These conditions satisfy the premises of definition A.1.

**Remark** (functionality versus function). In its programming sense, the term “function” will not be used here. A mathematical functionality differs from a software function; functionalities lack arguments. By virtue of its calling protocol, a programming function is effectively a class of functionalities.

**4.2. Procedure.** Procedures are sequences in a finite set of functionalities:

**Definition 4.3.** Let \( \langle \Psi, \Phi \rangle \) be a basis. A finite subset \( \mathcal{F} \subseteq \prod \Phi^{\prod \Psi} \) is a **catalog** of functionality.

**Definition 4.4.** Let \( \langle \Psi, \Phi \rangle \) be a basis and \( \mathcal{F} \subseteq \prod \Phi^{\prod \Psi} \) be a catalog of functionality. A **procedure** is a sequence \( \{f_n\} : \mathbb{N} \to \mathcal{F} \).

After noting the functionality’s successful generalization of the frame, the next question is whether the procedure correspondingly abstracts the process. We find that not all processes are “computable” as procedures based on a finite number of functionalities.

**4.3. Covering.** The relation holding between frame \( f \in \prod \Psi \times \prod \Phi \) and functionality \( f : \prod \Psi \to \prod \Phi \) is membership: either \( f \in f \) or \( f \not\in f \).

**Definition 4.5.** Let \( f \) be a frame and \( f \) be a functionality. The functionality **covers** the frame if \( f \in f \) (that is, \( f = (\psi, \phi) = (\psi, f(\psi)) \)).
Definition 5.2. Let \( \{ f_i \} \) be a sequence of frames and \( \{ f_n \} \) be a procedure. The procedure covers the sequence of frames if \( f_i \in f_i \) for each \( i \geq 1 \) (that is, \( f_i = (\psi_i, \phi_i) = (\psi_i, f_i(\psi_i)) \)).

Any procedure covers some process.

Theorem 4.7. Let \( (\Psi, \Phi) \) be a basis with persistent-volatile partition \( \Psi = \Phi \Xi \) and catalog of functionality \( \mathcal{F} \). Suppose \( \{ f_n \} : \mathbb{N} \rightarrow \mathcal{F} \) is a procedure. For each choice of persistent state \( \varphi \in \prod \Phi \) and volatile excitation sequence \( \{ \xi_n \} \in (\prod \Xi)^\mathbb{N} \) there is a process \( \{ f_n \} : \mathbb{N} \rightarrow \prod \Psi \times \prod \Phi \) such that the procedure covers the process.

Proof. The persistent state \( \varphi \) and volatile excitation \( \{ \xi_n \} \) are given. Inductively define sequence \( \{ \phi_n \} \) using base clause \( \phi_1 = \varphi \) and recursive clause \( \phi_{i+1} = f_i(\phi_i \xi_i) \) for \( i \geq 1 \) (dyadic notation, definition A.16ff).

We first show that the sequence \( \{ \phi_n \} \) lies in \( \prod \Phi \). Through \( \varphi \), definition 3.7 provides that \( \phi_1 \in \prod \Phi \). For the inductive part, the hypothesis \( \phi_i \in \prod \Phi \) implies that \( \phi_i \xi_i \in \prod \Psi \) by Theorem A.20. Since \( f_i \) maps \( \prod \Psi \) to \( \prod \Phi \), then \( f_i(\phi_i \xi_i) = \phi_{i+1} \in \prod \Phi \). This chain of implications concludes that \( (\phi_n \in \prod \Phi) \Rightarrow (\phi_{i+1} \in \prod \Phi) \).

Use \( \{ \phi_n \} \) and \( \{ \xi_n \} \) to define another sequence \( \{ f_n \} \) by setting \( f_i = (\phi_i \xi_i, \phi_{i+1}) \). Since \( \phi_i \xi_i \in \prod \Psi \) and \( \phi_i \xi_i \in \prod \Phi \), then \( (\phi_i \xi_i, \phi_{i+1}) \in \prod \Psi \times \prod \Phi \) so \( \{ f_n \} \) is a sequence of frames. Note that \( f_i = (\phi_i \xi_i, \phi_{i+1}) \) and \( f_{i+1} = (\phi_{i+1} \xi_{i+1}, \phi_{i+2}) \). In this case, the sequence is successively conjoint (definitions 3.3 and 3.4) because \( \phi_{i+1} \xi_{i+1} \mid \text{dom } \Phi = \phi_{i+1} \). As a successively conjoint sequence of frames, \( \{ f_n \} \) is a process (definition 5.5).

Since \( f_i = (\phi_i \xi_i, \phi_{i+1}) \), and by construction \( \phi_{i+1} = f_i(\phi_i \xi_i) \), then \( f_i = (\phi_i \xi_i, f_i(\phi_i \xi_i)) \) and \( f_i \in f_i \). By definition 4.6 procedure \( \{ f_n \} \) covers process \( \{ f_n \} \).

\[ \square \]

4.4. Uncoverable process. Although any procedure does cover some process, some processes have no covering procedure. See §B

5. Automata

The algorithm is conceived as a method to solve problems using a network of mechanistic steps consisting of decisions and contingent actions. An automaton is a formal machine whose architecture of states and transitions concretizes some aspects of the algorithm. The deterministic finite automaton (DFA, see example in §D.1) is a simple structure describing transit-based behavior. However, the DFA leaves unexplained the working mechanism underlying transitions. The DFA’s definition can be modified to effect closer alignment with our notion of software algorithm. The result is the actuated automaton, which mechanizes logic using structure analogous to programming language. An informal analogy between an automaton and a programming language will be proposed at the end of this section.

5.1. Locus. A formalization of the algorithm’s stepwise network of decisions and actions requires some means of indicating one’s place in the overall method, to track what is current and what is next. We provide this in the form of a set of loci, which serve as labels for the “locations” implicit in a program or algorithm.

Definition 5.1. A catalog of loci is a finite non-empty set \( \Lambda \), each member \( \lambda \) of which is called a locus.

Definition 5.2. Let \( \Lambda \) be a catalog of loci. A path is a sequence \( \{ \lambda_n \} : \mathbb{N} \rightarrow \Lambda \).

5.2. Summary. We now identify the three fundamental P’s:

- A path is a sequence in the catalog of loci \( \Lambda \).
- A process is a conjoint sequence in the frame space \( F = \prod \Psi \times \prod \Phi \).
- A procedure is a covering sequence in the catalog of functionality \( \mathcal{F} \).
5.3. **Auxiliary mechanisms.**

**Definition 5.3.** Let $\mathcal{F}$ be a catalog of functionality on basis $\langle \Psi, \Phi \rangle$. An *actuator* $a$ is a mapping $a: \prod \Psi \rightarrow \mathcal{F}$ from the process stimulus space to the catalog of functionality.

In other words, any $a \in \mathcal{F} \prod \Psi$ is an actuator.

**Definition 5.4.** A catalog of *actuation* is a non-empty finite set $A \subseteq \mathcal{F} \prod \Psi$ of actuators.

With each locus is associated exactly one designated actuator:

**Definition 5.5.** Let the *locator* be a surjective mapping $\ell: \Lambda \rightarrow A$.

To permit each actuator to be located, it is prerequisite that $|A| \leq |\Lambda|$ (the number of actuators is less than or equal to the number of loci).

**Definition 5.6.** Let $\Lambda$ be a catalog of loci. The *jump* function is a mapping $\Delta: \Lambda \times \prod \Psi \rightarrow \Lambda$.

5.4. **Actuated automaton.**

**Definition 5.7.** The structure $\mathcal{A} = \langle \Psi, \Phi, \mathcal{F}, A, \Lambda, \ell, \Delta \rangle$ of an *actuated* automaton consists of seven synchronized catalogs: a basis $\langle \Psi, \Phi \rangle$ with catalog of functionality $\mathcal{F}$, catalog of actuation $A$, catalog of loci $\Lambda$, locator function $\ell$, and jump function $\Delta$.

**Terminology.** Automata exist in many varieties. Since the actuated automaton occupies the entire present scope of interest, we forgo mandatory use of the qualifier “actuated.”

5.5. **Programming language analogy.** The automaton $\langle \Psi, \Phi, \mathcal{F}, A, \Lambda, \ell, \Delta \rangle$ of definition 5.7 resembles an algorithm written in an elementary programming language. The following comprise the analogy:

- the generating ensemble $\Phi$ of the persistent state space represents ordinary program variables;
- the remainder ensemble $\Psi \setminus \Phi$ represents sensory (volatile) external inputs;
- functionalities of the catalog $\mathcal{F}$ represent blocks of program assignment statements;
- actuators in $A$ implement if-then-elsif-else decisions,
- the jump function $\Delta$ is a “goto” indicating the next point of execution, and
- loci in $\Lambda$ are labels serving as “goto-able” points of execution.

6. **Iteration**

While leaving undefined the notion of a step in an algorithm, we do formalize it for the automaton. Identifying a step space leads to defining iterative operators, and hence to iteration.

6.1. **Step space.** The step space underlies automata. It is formed by augmenting a catalog of loci to the building blocks of processes and procedures, namely frames and functionalities.

**Definition 6.1.** Suppose $\Lambda$ is a catalog of loci. Let basis $\langle \Psi, \Phi \rangle$ underly frame space $F = \prod \Psi \times \prod \Phi$ and catalog of functionality $\mathcal{F} \subseteq \prod \Phi \prod \Psi$. A *step space* $S$ is the Cartesian product $S = \Lambda \times \mathcal{F} \times F$.

**Remark.** The volatile volatile excitation, whose generating ensemble is $\Psi \setminus \Phi$, is intrinsically part of the definition of step space. Lest this implicit fact be forgotten, we shall adopt explicit but redundant notation as reminders.

**Definition 6.2.** Let $s = (\lambda, f, f)$ be a member of step space $S = \Lambda \times \mathcal{F} \times F$. Define the *locus* projection $\tilde{\Omega}_\Lambda: S \rightarrow \Lambda$ by setting $\tilde{\Omega}_\Lambda(\lambda, f, f) = \lambda$. Similarly define the *frame* and *functionality* projections by $\tilde{\Omega}_\mathcal{F}(\lambda, f, f) = f$ and $\tilde{\Omega}_\Psi(\lambda, f, f) = f$ respectively.
Definition 6.3. Let \( s = (\lambda, f, f) \) be a member of step space \( S = \Lambda \times \mathcal{F} \times F \). The step \( s \) is consistent if \( \overline{\Omega}_F(s) = f \in f = \overline{\Omega}_\mathcal{F}(s) \) (that is, its frame is a member of its functionality).

6.1.1. Sequence projection.

Definition 6.4. Let \( S = \Lambda \times \mathcal{F} \times F \) be a step space and \( \mathcal{F} \) be a denumerable index set. A walk is a sequence \( \mathcal{F} \to S \) of steps (usually \( \mathcal{F} \) will be the natural numbers \( \mathbb{N} \)).

Remark. We revisit the three fundamental P’s (path, procedure, and process – \[6.2\]). A walk in step space decomposes into these three sequences: walk \( i = (\text{path}_i, \text{procedure}_i, \text{process}_i) \). This triple is not logically independent; being so, shorter characterizations of step space exist. However we retain the present representation, favoring its three-element formulation.

Definition 6.5. [Extended projection] Let \( \{s_n\} \) be a walk in step space \( S = \Lambda \times \mathcal{F} \times F \) and let \( \mathcal{F} \) be a denumerable index set (usually \( \mathbb{N} \)). Use the locus projection \( \overline{\Omega}_\Lambda : S \to \Lambda \) of definition \[6.2\] to construct the sequential path projection \( \overline{\Omega}_\Lambda : S^\mathcal{F} \to \Lambda^\mathcal{F} \) via setting \( \overline{\Omega}_\Lambda(\{s_n\}) = \{(i, \overline{\Omega}_\Lambda(s_i)): (i, s) \in \{s_n\}\} \). Similarly define the sequential process and procedure projections \( \overline{\Omega}_F : S^\mathcal{F} \to F^\mathcal{F} \) and \( \overline{\Omega}_\mathcal{F} : S^\mathcal{F} \to \mathcal{F}^\mathcal{F} \).

Notation. With \( \{x_n\}_\mathcal{F} \to X \) a sequence in some set \( X \), we alternatively denote the sequence’s \( i \)th term by \( x_i = \{x_n\}_\mathcal{F}(i) \).

Lemma 6.6. Let \( \langle \Psi, \Phi \rangle \) and locus set \( \Lambda \) be the bases for step space \( S = \Lambda \times \mathcal{F} \times F \). Let \( \{s_n\} \) be a walk \( \mathcal{F} \to S \). For each \( i \in \mathcal{F} \), \( \overline{\Omega}_\Lambda(\{s_n\})(i) = \overline{\Omega}_\Lambda(\{s_n\})(i) \); that is, the \( i \)th term of the sequential path projection equals the locus projection of the \( i \)th step.

Analogous assertions are true of the remaining sequential projections: \( \overline{\Omega}_F(\{s_n\})(i) = \overline{\Omega}_F(\{s_n\})(i) \) and \( \overline{\Omega}_{\mathcal{F}}(\{s_n\})(i) = \overline{\Omega}_{\mathcal{F}}(\{s_n\})(i) \).

Proof. By definition \[6.5\] the sequential path projection \( \overline{\Omega}_\Lambda : S^\mathcal{F} \to \Lambda^\mathcal{F} \) is \( \overline{\Omega}_\Lambda(\{s_n\}) = \{(i, \overline{\Omega}_\Lambda(s_i)): (i, s) \in \{s_n\}\} \). The \( i \)th term of \( \overline{\Omega}_\Lambda(\{s_n\}) \) is \( (\overline{\Omega}_\Lambda(\{s_n\}))(i) \). From set builder notation we observe that the \( i \)th term of the expression \( \{(i, \overline{\Omega}_\Lambda(s_i)): (i, s) \in \{s_n\}\} \) is \( \overline{\Omega}_\Lambda(s_i) = \overline{\Omega}_\Lambda(\{s_n\})(i) \). Since the sequences are equal, then each of their corresponding terms are equal: \( \overline{\Omega}_\Lambda(\{s_n\})(i) = \overline{\Omega}_\Lambda(\{s_n\})(i) \). Demonstration is similar for the other two sequential projections.

6.2. Iterative operators.

Definition 6.7. Let \( S \) be a step space with basis \( \langle \Psi, \Phi \rangle \). Suppose the volatile volatile excitation space \( \Psi \setminus \Phi \) is non-empty. A iterative operator is a mapping \( V : S \to S \).

6.2.1. Disambiguation. An iterative operator maps a step space into itself. One element of a step space is a reactive state space, having persistent and volatile components. A functionality maps a reactive state space into the persistent subset of itself.

6.2.2. Walk of iterative operator.

Definition 6.8. Let \( \langle \Psi, \Phi \rangle \) be a basis with step space \( S = \Lambda \times \mathcal{F} \times (\prod \Psi \times \prod \Phi) \). Suppose \( V : S \to S \) is an iterative operator, step \( s \in S \), and \( \langle \xi_n \rangle \in (\prod \Psi \times \prod \Phi) \) is an iterative operator. Define inductively a sequence of steps by setting \( s_1 = s \) and \( s_{i+1} = V(s_i) \) for each \( i \geq 1 \). The walk of \( s \in S \) under \( V \), assuming sequence of volatile excitation \( \langle \xi_n \rangle \), is the walk \( \{s_n\} \).

Terminology. An iterative operator’s \( i \)th iteration is its walk’s \( (i + 1) \)th term.
6.3. **Automaton-induced iterative operators.** Iteration of automata guarantees properties not necessarily enjoyed by other classes of iterative operators: automata generate consistent steps having conjoint processes.

6.3.1. **Automaton as transformation.**

**Definition 6.9.** Let \( \langle \Psi, \Phi \rangle \) be a basis with persistent-volatile partition \( \Psi = \Phi \Xi \) and step space \( S = \Lambda \times \mathcal{F} \times (\prod \Psi \times \prod \Phi) \). Let \( \mathcal{A} = \langle \Psi, \Phi, \mathcal{F}, \Lambda, \ell, \Delta \rangle \) be an actuated automaton. Let \( \xi \in \prod \Xi \) be an event stimulus and \( (\lambda, f, f) = (\lambda, f, (\psi, \phi)) \in S \) be a step. The transform \( T_{\mathcal{A}} \) induced by \( \mathcal{A} \) is

\[
(\lambda, f, (\psi, \phi)) \xrightarrow{T_{\mathcal{A}}} (\lambda', f', (\psi', \phi'))
\]

where

\[
\begin{align*}
\lambda' &= \Delta(\lambda, \psi), \\
f' &= (\ell(\Delta(\lambda, \psi)))(f(\psi)\xi'), \\
f' &= (\psi', \phi', \ell(\psi, f(\psi)\xi')).
\end{align*}
\]

**Remark.** A partial unfolding of these expressions’ generators clarifies the roles of components in overall mechanism:

1. Current reactive state is \( \psi = \phi\xi \).
2. Current locus is \( \lambda \).
3. Current actuator is \( a = \ell(\lambda) \).
4. Current functionality is \( f = a(\psi) = (\ell(\lambda))(\psi) \).
5. Current frame is \( f = (\psi, f(\psi)) \).
6. Current step is \( s = (\lambda, (\psi, (\ell(\lambda))(\psi), f(\psi))) \).
7. Next reactive state is \( \psi' = \phi'\xi' = f(\psi)\xi' \) (by conjointness).
8. Next locus is found through the jump function: \( \lambda' = \Delta(\lambda, \psi) \).
9. Next actuator is \( a' = \ell(\lambda') \).
10. \( a' = \ell(\Delta(\lambda, \psi)) \).
11. Next functionality is \( f' = a'(\psi') \).
12. \( f' = (\ell(\Delta(\lambda, \psi)))(\psi') \).
13. \( f' = (\ell(\Delta(\lambda, \psi)))(f(\psi)\xi') \).
14. Next frame is \( f' = (\psi', f'(\psi')) \).
15. \( f' = (f(\psi)\xi', f'(f(\psi)\xi')) \).
16. \( f' = (f(\psi)\xi', (\ell(\Delta(\lambda, \psi)))(f(\psi)\xi'))(f(\psi)\xi')) \).
17. Next step is \( s' = (\Delta(\lambda, \psi), a'(\psi'), (\psi', f'(\psi'))) \).

6.3.2. **Automaton as iterative operator.**

**Theorem 6.10.** Let \( \langle \Psi, \Phi \rangle \) be a basis with persistent-volatile partition \( \Psi = \Phi \Xi \) and step space \( S \). Let \( \mathcal{A} \) be an automaton. The transform \( T_{\mathcal{A}} : S \rightarrow S \) induced by \( \mathcal{A} \) is an iterative operator (that is, if \( s \in S \), then \( T_{\mathcal{A}}(s) \in S \)).

**Proof.** By definition 5.7, automaton \( \mathcal{A} \) consists of components \( \langle \Psi, \Phi, \mathcal{F}, \Lambda, \ell, \Delta \rangle \). Other premises are that the iterative operator’s domain is \( S \), the persistent-volatile partition \( \Psi = \Phi \Xi \), and that \( \xi \in \prod \Xi \).

Suppose \( \xi \in \prod \Xi \) and \( s \in S \). By definition 6.1 of step space, there exist locus \( \lambda \in \Lambda \), functionality \( f \in \mathcal{F} \subseteq \prod \Phi \Pi \Psi \), and frame \( f = (\psi, \phi) \in \prod \Psi \times \prod \Phi \) such that \( s = (\lambda, (\psi, \phi), f) \).

Definition 6.2 calls for application \( (\ell(\lambda))(\phi) \) to define succeeding functionality \( f' \). Definition 5.3 specifies \( \ell : \Lambda \rightarrow \mathcal{A} \), so \( a = \ell(\lambda) \in \mathcal{A} \) is an actuator. Definition 5.3 specifies \( a : \prod \Phi \rightarrow \mathcal{F} \), so application \( f' = (\ell(\lambda))(\phi) = a(\phi) \) is a functionality in \( \mathcal{F} \).
Definition 6.9 next calls for evaluating \((\psi', \phi') = (\phi \xi, f'(\phi \xi))\) as the succeeding frame \(f'\). By premise \(\xi \in \prod \Xi\). By virtue of its origin as a frame ordinate, \(\phi \in \prod \Phi\). Since \(\Psi = \Phi \Xi\), then \(\phi \xi\) is a valid dyadic product and \(\psi' = \phi \xi \in \prod \Psi\). Since definition 4.3 asserts \(\mathcal{F} \subseteq \prod \Phi \prod \Psi\) and \(f' \in \mathcal{F}\), then \(f' : \prod \Psi \to \prod \Phi\). With \(\phi \xi \in \prod \Psi\), then \(\phi' = f'(\phi \xi) \in \prod \Phi\). Hence \(f' = (\psi', \phi') \in \prod \Psi \times \prod \Phi\), and \(f'\) is a frame.

Finally, definition 6.9 calls for the succeeding locus as \(\lambda' = \Delta(\lambda, \psi)\). Definition 5.9 specifies the jump function as a mapping \(\Delta : \Lambda \times \prod \Psi \to \Lambda\). It is established above that \(\lambda \in \Lambda\) and \(\psi \in \prod \Psi\), so \(\lambda' = \Delta(\lambda, \psi) \in \Lambda\) is a locus.

With locus \(\lambda' \in \Lambda\), functionality \(f' \in \mathcal{F}\), and frame \(f' = (\psi', \phi') \in \prod \Psi \times \prod \Phi\), we then summarize that \(s' = (\lambda', f', (\psi', \phi')) \in S\) is a step, and conclude that transform \(T_\mathfrak{A}\) is an iterative operator \(S \to S\).

**Notation.** Application of the iterative operator \(T_\mathfrak{A}\) induced by automaton \(\mathfrak{A}\) is denoted \(s' = \mathfrak{A}(s)\).

**Notation.** Let \(S\) be a step space with automaton \(\mathfrak{A}\) inducing iterative operator \(T_\mathfrak{A}\). The walk of \(s \in S\) under \(T_\mathfrak{A}\) assuming sequence of volatile excitation \(\{\xi_n\}\) is denoted \(\mathfrak{A}^N_{\{\xi_n\}}(s)\).

**Remark.** The notation \(\mathfrak{A}^N_{\{\xi_n\}}(s)\) is a reminder of the important role of the sequence of volatile excitations \(\{\xi_n\}\). While each \(\xi\) is entirely determined by initial frame \(\psi\) within step \(s\), this notation emphasizes that the volatile excitations are essentially free variables. Persistent variables are bound.

### 6.3.3. Automaton iterative properties.

**Theorem 6.11.** Let \(\mathfrak{A}\) be an automaton and \(S\) be a step space with persistent-volatile partition \(\Psi = \Phi \Xi\). Suppose step \(s \in S\) and event \(\xi \in \Xi\). Frame \(\check{U}_\Phi(s)\) conjoints frame \(\mathfrak{A}_\xi(\check{U}_\Phi(s))\).

**Proof.** By hypothesis \(\xi \in \prod \Xi\) and \(s \in S\). By definition 6.1 of step space, there exist locus \(\lambda \in \Lambda\), frame \(f = (\psi, \phi) \in \prod \Psi \times \prod \Phi\), and functionality \(f \in \mathcal{F}\) such that \(s = (\lambda, f, (\psi, \phi))\). Definition 6.2 establishes that \(\check{U}_\Phi(s) = (\psi, \phi)\).

By Theorem 6.10, the automaton induces an iterative operator, so there exists \(\mathfrak{A}_\xi(\check{U}_\Phi(s)) = (\lambda', f', (\psi', \phi')) \in S\). Again by definition 6.2, \(\check{U}_\Phi(\mathfrak{A}_\xi(\check{U}_\Phi(s))) = (\psi', \phi')\).

Definition 6.9 evaluates \(f' = (\psi', \phi') = (\phi \xi, f'(\phi \xi))\) as the succeeding frame. Definition 3.3 asserts that frame \((\psi, \phi)\) conjoints frame \((\psi', \phi')\) if \(\psi' \mid \text{dom} \Phi = \phi\). Here \(\psi' = \phi \xi\), so \(\phi \xi \mid \text{dom} \Phi = \phi\) by virtue of persistent-volatile partition \(\Psi = \Phi \Xi\). Thus we conclude \((\psi, \phi)\) conjoints \((\psi', \phi')\).

Since \(\check{U}_\Phi(s) = (\psi, \phi), (\psi, \phi)\) conjoints \((\psi', \phi'), (\psi', \phi') = \check{U}_\Phi(\mathfrak{A}_\xi(s))\), then by transitivity \(\check{U}_\Phi(s)\) conjoints \(\check{U}_\Phi(\mathfrak{A}_\xi(s))\).

**Theorem 6.12.** Let \(\mathfrak{A}\) be an automaton and \(S\) be a step space with step \(s \in S\) and volatile excitation \(\{\xi_n\}\). Sequential process projection \(\check{U}_\Phi(\mathfrak{A}^N_{\{\xi_n\}}(s))\) is indeed a process per definition 3.3.

**Proof.** Definition 3.3 asserts that a process is a successively conjoint sequence of frames. To show contradiction, hypothesize that the frame sequence \(\check{U}_\Phi(\mathfrak{A}^N_{\{\xi_n\}}(s))\) is not a process. Then there is some index \(i\) such that frame \(\check{U}_\Phi(s_i)\) does not conjoint frame \(\check{U}_\Phi(s_{i+1})\).

Let \(s_i\) be the \(i\)th step of walk \(\mathfrak{A}^N_{\{\xi_n\}}(s)\). From definition 6.8, the succeeding step is \(s_{i+1} = \mathfrak{A}_\xi(s_i)\). By Theorem 6.11, frame \(\check{U}_\Phi(s_i)\) conjoints frame \(\check{U}_\Phi(\mathfrak{A}_\xi(s_i))\). This contradicts the conclusion drawn from the hypothesis that the frame sequence is not a process, so \(\check{U}_\Phi(\mathfrak{A}^N_{\{\xi_n\}}(s))\) is a process.

Be a step consistent or not, in automaton-based iteration that step’s successor is consistent.

**Theorem 6.13.** Let \(\mathfrak{A}\) be an automaton and \(S\) be a step space with persistent-volatile partition \(\Psi = \Phi \Xi\). Suppose step \(s \in S\) and event \(\xi \in \Xi\). Step \(\mathfrak{A}_\xi(s)\) is consistent.
Proof. By hypothesis $\xi \in \prod \Xi$ and $s \in S$. By definition 6.1 of step space, there exist locus $\lambda \in \Lambda$, frame $f = (\psi, \phi) \in \prod \Psi \times \prod \Phi$, and functionality $f \in \mathcal{F}$ such that $s = (\lambda, (\psi, \phi), f)$.

By Theorem 6.10 the automaton induces an iterative operator, so there exists $\aleph_{\xi}(s) = (\lambda', (\psi', \phi'), f') \in S$.

By definition 6.2 $\aleph_{\mathcal{F}}(\aleph_{\xi}(s)) = f' = (\psi', \phi')$ and $\aleph_{\mathcal{F}}(\aleph_{\xi}(s)) = f'$.

Definition 6.9 evaluates $f' = (\psi', \phi') = (\phi \xi, f'(\phi \xi))$ as the succeeding frame. This complex relation separates into simple conditions $\psi' = \phi \xi$ and $\phi' = f'(\phi \xi) = f'(\psi')$. The assertion $f' = (\psi', \phi') = (\psi', f'(\psi'))$ bears the same meaning as $f' = (\psi', \phi') \in f'$.

Definition 6.3 states that step $\aleph_{\xi}(s)$ is consistent if $\aleph_{\mathcal{F}}(\aleph_{\xi}(s)) = f' \in f' = \aleph_{\mathcal{F}}(\aleph_{\xi}(s))$, which is here satisfied.

Theorem 6.14. Let $\aleph$ be an automaton and $S$ be a step space with consistent step $s \in S$ and volatile excitation $\{\xi_n\}$. Suppose $\{s_n\}$ is the walk $\aleph^{N}_{\{\xi_n\}}(s)$. Sequential procedure projection $\mathcal{F}_{\aleph}(\{s_n\})$ covers sequential process projection $\mathcal{F}_{\aleph}^{N}(\{s_n\})$.

Proof. We have the premises that step $s \in S$ is consistent, that $\{\xi_n\}$ is an excitation, that $\aleph$ is an automaton, and that $\{s_n\}$ is the walk $\aleph^{N}_{\{\xi_n\}}(s)$. We temporarily suppress the repetitive lengthy expression $\aleph^{N}_{\{\xi_n\}}(s)$ through the abbreviation $\aleph^{N}$.

Induction demonstrates that each step of walk $\aleph^{N}$ is consistent. For the base clause, the case $i = 1$ is true by hypothesis, since the initial step $\aleph^{N}(1) = s$ is presumed consistent. For the recursive clause, definition 6.8 provides that $s_{i+1} = \aleph(\xi_i, s_i)$ for each $i \geq 1$. By theorem 6.13 step $\aleph(\xi_i, s_i) = \aleph^{N}(i + 1)$ is consistent. By the axiom of induction, step $\aleph^{N}(i)$ is consistent for each $i \geq 1$.

By definition 6.3 and the conclusion that step $\aleph^{N}(i)$ is consistent for each $i \geq 1$, it follows that $\aleph_{\mathcal{F}}(\aleph^{N}(i)) \in \aleph_{\mathcal{F}}(\aleph^{N}(i))$, also for each $i \geq 1$.

Substituting $\aleph^{N} = \{s_n\}$ into lemma 6.6 yields frame $(\mathcal{F}_{\aleph}(\aleph^{N}))(i) = \aleph_{\mathcal{F}}(\aleph^{N}(i))$ and functionality $(\mathcal{F}_{\aleph}(\aleph^{N}))(i) = \aleph_{\mathcal{F}}(\aleph^{N}(i))$. From equality it then follows that $\aleph_{\mathcal{F}}^{N}(i) \in (\mathcal{F}_{\aleph}(\aleph^{N}))(i)$ for each $i \geq 1$. In simple language, the $i^{th}$ term of the process projection is a member of the $i^{th}$ term of the procedure projection. This satisfies the requirement of definition 4.6 that the procedure covers the sequence of frames: $f_i \in f_i$ for each $i \geq 1$.

7. Reverse inference

The construction of automata provides that steps unfold in sequential fashion – that is, the next step becomes known after completing the current step. Consequently automata inherit an intrinsic “forward” orientation. It is also reasonable to inquire what may have occurred in previous steps. This question is the motivation for reverse inference, which considers automata operating backwards.

7.1. Iterative converse. Let $(\Psi, \Phi)$ be a basis with persistent-volatile partition $\Psi = \Phi \Xi$ and step space $S = \Lambda \times \mathcal{F} \times \prod \Psi \times \prod \Phi$. Suppose step $s = (\lambda, f, (\psi, \phi)) \in S$, whence the volatile excitation of $s$ is $\xi = \psi | \text{dom } \Xi$.

Definition 7.1. Let $(\Psi, \Phi)$ be a basis with step space $S$ and iterative operator $V : S \rightarrow S$. The iterative converse of $V$ is the mapping $\bar{V} : S \rightarrow \mathcal{P}(S)$ defined by

$$\bar{V}(s) = \{ \hat{s} \in S : V(\hat{s}) = s \},$$

where $\mathcal{P}(S)$ denotes the power set of $S$ (in other words, the iterative converse is a mapping from a step to a set of steps).
Finally, substituting

\[ \hat{\mathcal{A}}(s) = \{ \hat{s} \in \mathbb{S} : \mathcal{A}(\hat{s}) = s \}. \]

**Remark.** Although referring informally to the converse \( \hat{\mathcal{A}} \) of an automaton \( \mathcal{A} \), speaking precisely we have defined the converse \( \hat{s} \) of a step \( s \in \mathbb{S} \) within the step space associated with that automaton.

A system of equations ensues with subscripted variables known and held constant, and unsubscripted variables free. Roots of the system represent discrete solutions. Let us look carefully at the further case that

\[ s = (\lambda, f, f) = (\lambda, f, (\psi, \phi)) \in \Lambda \times \mathcal{A} \times \mathcal{F}. \]

\[ \hat{\mathcal{A}}(\lambda_0, f_0, f_0) = \{ (\lambda, f, f) \in \mathbb{S} : \mathcal{A}(\lambda, f, f) = (\lambda_0, f_0, f_0) \} \]

\[ \hat{\mathcal{A}}(\lambda_0, f_0, (\psi_0, \phi_0)) = \{ (\lambda, f, (\psi, \phi)) \in \mathbb{S} : \mathcal{A}(\lambda, f, (\psi, \phi)) = (\lambda_0, f_0, (\psi_0, \phi_0)) \}. \]

**7.3. Constraining equations.** State transition in an actuated automaton is built in three successive phases: locus state, functionality state, and frame state.

**Definition 6.9** presents rules governing forward state transition in the form of three equations, portraying current state as known and unknown future state as uniquely determined by formulas. This sense can be reversed, with current state known and feasible past states represented as unknowns.

The automaton-induced forward transformation \( \mathcal{A} : \mathbb{S} \rightarrow \mathbb{S} \) has been set (definition 6.9) as

\[ \lambda' = \Delta(\lambda, \psi), \]

\[ f' = (\ell(\Delta(\lambda, \psi)))(f(\psi)\xi'), \]

\[ f' = (\psi', \phi') = ([f(\psi)\xi', [[\ell(\Delta(\lambda, \psi))](f(\psi)\xi')]][[f(\psi)\xi'])]. \]

The respective governing backwards transformations are

\[ \lambda_0 = \Delta(\lambda, \psi), \]

\[ f_0 = (\ell(\Delta(\lambda, \psi)))(\psi_0), \]

\[ f_0 = (\psi_0, \phi_0) = ([f(\psi)\xi_0], [[\ell(\Delta(\lambda, \psi))](f(\psi)\xi_0)][[f(\psi)\xi_0])]. \]

Due to conjunction, \( \psi_0 = f(\psi)\xi_0 = \phi_0\xi_0 \) throughout:

\[ \lambda_0 = \Delta(\lambda, \psi), \]

\[ f_0 = (\ell(\Delta(\lambda, \psi)))(\psi_0), \]

\[ f_0 = (\psi_0, \phi_0) = ([f(\psi)\xi_0], [[\ell(\Delta(\lambda, \psi))](\psi_0)][(\psi_0)]]). \]

Substituting \( \lambda_0 = \Delta(\lambda, \psi) \) and \( a_0 = \ell(\lambda_0) \) into the last two formulas,

\[ \lambda_0 = \Delta(\lambda, \psi), \]

\[ f_0 = (\ell(\lambda_0))(\psi_0) = a_0(\psi_0), \]

\[ f_0 = (\psi_0, \phi_0) = ([f(\lambda_0)](\psi_0)[(\psi_0)]) = (\psi_0, [a_0(\psi_0)](\psi_0)). \]

Finally, substituting \( f_0 = a_0(\psi_0) \) into the last two formulas,

\[ \lambda_0 = \Delta(\lambda, \psi), \]

\[ f_0 = f_0, \]

\[ f_0 = (\psi_0, f_0(\psi_0)) = f_0. \]
Only the first of these is a “real” constraint; the others are identities.

7.4. **Solution set.** A feasible set for an equation is the collection of values that satisfy the equation. Producing the solution for a system of constraining equations involves intersecting the individual feasible sets $Q_i$. The general solution set for the system of equations is $Q = \bigcap_i Q_i$.

The case of the converse automaton $\tilde{A}$ consists of three constraining equations (§7.3). First of these is $\lambda_0 = \Delta(\lambda, \psi)$. Let $S = \Lambda \times \mathcal{F} \times (\prod \Psi \times \prod \Phi)$ be a fully elaborated step space. In set-builder notation the feasible set is $Q_1 = \{(\lambda, f, (\psi, \phi)) \in S : \lambda_0 = \Delta(\lambda, \psi)\}$. The other constrains are identities, which all members of $S$ satisfy—in other words, $Q_2 = Q_3 = S$. The general solution is $Q = Q_1 \cap Q_2 \cap Q_3 = Q_1 \cap S \cap S = Q_1$, or $Q = \{(\lambda, f, (\psi, \phi)) \in S : \lambda_0 = \Delta(\lambda, \psi)\}$. An understandable abuse of terminology says that the solution set is $\lambda_0 = \Delta(\lambda, \psi)$, which is technically a constraining equation.

8. **Cone**

A cone is a construct prepared with the iterative converse of an actuated automaton. It consists of all finite backwards walks converging to a given point. The term “cone” is more ideologic than geometric.

8.1. **Description.** The actuated automaton possesses a non-deterministic\(^4\) converse relation. See §7.1.

A collection of reverse walks is realized through repetitive re-application of the converse, converging to a designated *crux* step. These iterative chains may be localized (trimmed to finite length) by enforcing some stopping criterion. This construction results in the cone, a structured set of possible localized walks eventually leading to the crux step. The starting points of such walks are known as *precursor* steps of the crux step.

8.2. **Inductive generation.**

**Definition 8.1.** Let $S$ be a step space containing crux step $s_{crux}$. Suppose $V : S \rightarrow S$ is an iterative operator with converse $\tilde{V} : S \rightarrow \mathcal{P}(S)$.

(base clause) Let base protoset $G^{(0)}_{\mathcal{O}} = \{s_{crux}\}$ be the $0^{th}$ predecessor generation of $s_{crux}$.

(inductive clause) The $(n + 1)^{th}$ generation predecessors are defined in terms of the $n^{th}$ generation:

$$G^{(n+1)}_{\mathcal{O}} = \bigcup_{s \in G^{(n)}_{\mathcal{O}}} \tilde{V}(s).$$

**Remark.** This definition places protoset $G^{(1)}_{\mathcal{O}} = \tilde{V}(s_{crux})$.

**Remark.** For a discussion of protoset $G^{(n)}_{\mathcal{O}}$ in context of the Cartesian product, see Appendix §A.7.

8.3. **Partial order.** Membership in a converse iterative operator induces a partial ordering:

**Definition 8.2.** Let $S$ be a step space with converse iterative operator $\tilde{V} : S \rightarrow \mathcal{P}(S)$. If $s' \in \tilde{V}(s)$, then $s'$ precedes $s$, written $s' \prec s$.

\(^4\)That is, the *converse* is not generally a pointwise invertible mapping as suggested by the term *inverse*. 
8.4. **Predecessor walk.** A predecessor walk begins at step \( s_0 = s_{\text{crux}} \) and proceeds backwards, indexing through the negative integers. In this case we abuse the proper sense of the term “sequence” by permitting an indexing not being the natural numbers.

**Definition 8.3.** Let \( \mathbb{S} \) be a step space. A localized predecessor walk starting with step \( s_0 = s_{\text{crux}} \) is a \textit{finite} sequence in step space such that \( s_{i-1} \prec s_i \) for every \( i \leq 0 \).

**Remark.** For example in the case \( i = -2 \), we have \( s_{-3} \prec s_{-2} \).

Since a localized predecessor walk \( w \) is a \textit{finite} sequence of steps, then it has a finite number of terms which run in index \( i \) from \(- (n - 1) \leq i \leq 0\), where \( n = |w| < \infty \) is the number of steps in \( w \).

**Definition 8.4.** A set \( \mathbb{W} \) of localized predecessor walks, all starting at \( w_0 = s_{\text{crux}} \), is complete if

\[
\forall (w \in \mathbb{W}) \forall (-(|w| - 2) \leq i \leq 0) \forall (s \in \hat{V}(w_i)) \exists (e \in \mathbb{W}) : w_i = e_i \land s = e_{i-1}.
\]

**Remark.** Completeness assures combinatorial diversity. An algorithm equivalent to the above one-liner is:

```plaintext
for each \( w \) in \( \mathbb{W} \) begin
    # number of steps in localized walk is \( |w| \)
    # abused index runs between 0 [for start step \( w_{\text{crux}} \)] and \(- (|w| - 1) \) [last predecessor step]
    # no iteration through \( i = - (|w| - 1) \) because then \( s \equiv e_{i-1} \) below would be undefined
    for \( i = 0 \) downto \(- (|w| - 2) \) begin
        for each \( s \) in \( \hat{V}(w_i) \) begin
            if there is a member \( e \in \mathbb{W} \) such that \( w_i = e_i \) and \( s = e_{i-1} \) then
                answer = TRUE
            else
                answer = FALSE
            if answer == FALSE then return FALSE
        end
    end
end
return TRUE
```

**Definition 8.5.** Let \( w \) and \( w' \) be localized predecessor walks starting at \( w_0 = w'_0 = s_{\text{crux}} \). Suppose the length of \( w \) is \( |w| = n \) and the length of \( w' \) is \( |w'| = m \), with \( m \leq n \). If \( w'(i) = w(i) \) for every \(- (m - 1) \leq i \leq 0 \), then \( w \) and \( w' \) are dependent with \( w' \) dispensable.

**Definition 8.6.** Let \( \mathbb{W} \) be a set of localized predecessor walks starting at \( s_0 = s_{\text{crux}} \). The set is independent if it contains no dispensable member.

8.5. **Cone.**

**Definition 8.7.** A cone \( C \) is a complete independent set of localized predecessor walks starting at \( s_{\text{crux}} \).

**Remark.** [Stopping rule] We avoid the specificity of various stopping criteria (§8.1) by introducing the equivalent but arbitrary notion of localization.

**Definition 8.8.** Let \( C \) be a cone with \( w \in C \) a member localized predecessor walk. Suppose \( n = |w| \) is the number of steps in \( w \). The terminus \( w(-(n-1)) = w_{-(n-1)} \) is the edge step of walk \( w \).

**Definition 8.9.** Let \( C \) be a cone. Its \textit{edge}, written \( \text{edge} \, C \), is the collection of edge steps of all member localized predecessor walks.

**Definition 8.10.** An acyclic cone has no cycle (loop) in its path projection \( \overrightarrow{\mathcal{T}}_A \) (see §6.5).
Theorem 8.11. The acyclic cone 𝐶 and edge 𝐶 are in one-to-one correspondence via the edge step relation of a localized predecessor walk.

Proof. Assume the opposite: there are different localized predecessor walks with the same edge step. Let 𝑢 and 𝑣 be two different walks with common edge step 𝑠common.

Suppose |𝑢| = 𝑚 and |𝑣| = 𝑛, so the indexes of 𝑠common are −(𝑚 − 1) and −(𝑛 − 1) respectively.

We assert that if 𝑢−(𝑚−1)+𝑖 = 𝑣−(𝑛−1)+𝑖 for some 0 ≤ 𝑖, then 𝑢−(𝑚−1)+(𝑖+1) = 𝑣−(𝑛−1)+(𝑖+1).

Suppose sequencing is governed by an actuated automaton 𝑂. So sequenced, the next step in predecessor walk 𝑢 is 𝑢−(𝑚−1)+(𝑖+1) = 𝑂(𝑢−(𝑚−1)+𝑖). Similarly, the next step in 𝑣 is 𝑣−(𝑛−1)+(𝑖+1) = 𝑂(𝑣−(𝑛−1)+𝑖).

But if 𝑢−(𝑚−1)+𝑖 = 𝑣−(𝑛−1)+𝑖, then 𝑂(𝑢−(𝑚−1)+𝑖) = 𝑂(𝑣−(𝑛−1)+𝑖) = 𝑂(𝑠). By transitivity of equality, 𝑢−(𝑚−1)+(𝑖+1) = 𝑣−(𝑛−1)+(𝑖+1).

Without loss of generality suppose 𝑚 ≤ 𝑛. Then 𝑢−(𝑚−1)+𝑖 = 𝑣−(𝑛−1)+𝑖 is true for 𝑖 = 0, 1, ..., 𝑚 − 1. At 𝑖 = 𝑚 − 1 we have 𝑠common = 𝑢0 = 𝑣−(𝑛−1)+(𝑖+1) = 𝑣−(𝑛−𝑚). Since 𝑣 is a localized predecessor walk of a cone, then 𝑣0 = 𝑠common. But 𝑣0 = 𝑠common = 𝑣−(𝑛−𝑚). Because the cone is assumed acyclic, 𝑣0 and 𝑣−(𝑛−𝑚) must then be the same identical step – that is, 𝑚 = 𝑛.

Here the assumption of two different local predecessor walks with the same edge step leads to the contradiction that both are indeed the same identical walk. This means local predecessor walks within an acyclic cone 𝐶 are in one-to-one correspondence with edge 𝐶 via the edge step relation.

9. Operational profile

An operational profile is a limit of the cumulative history of software execution ratios under normal operations, which is troublesome to define.

Remark. This section frequently uses the compound idiom that \{𝑥\}_𝑛 represents an anonymous sequence of objects of the same type as \(x\). That is, if \(X\) is the set of all \(x_i\), then \(\{x_n\} : \mathbb{N} \to X\).

9.1. Musa’s operational profile. Musa et al intended operational profiles as a tool for analysis of software reliability. A notion of the operational profile appeared in their pioneering exposition [3]. This reference gives a definition in terms of the program’s higher purpose, as reflected in run types. Consequently an operational profile is the set of run types that the program can execute along with the probabilities they will occur. One can easily envision its extension to smaller program units.

9.2. Extended operational profile. We shall extend run types into steps, the elementary quantum of automata. This detaches the operational profile concept from notions such as run types which are part of human understanding rather than algorithmic structure (however, the idea pops up elsewhere). Despite appearances, this extension is not so large – the only addition is a method for counting step events.

9.3. Counting. Let \(\{s_i\}\) be a walk (infinite sequence of steps) and let \(Z\) be an arbitrary reference set of steps (members of step space \(\mathbb{S} = \Lambda \times \mathcal{F} \times \mathcal{F}\)). Simply summarized, \(N_Z(\{s_n\}, k)\) denotes the number of occurrences of any member of \(Z\) before or at the \(k^{th}\) automaton step.

Details follow for those interested. When the \(i^{th}\) step of the walk is a member of \(Z (s_i \in Z)\), then \(\{s_n\}\) is said to arrive at \(i\).

Definition 9.1. An arrival function is a sequence \(\varphi : \{1, 2, \ldots\} \to \{n_1, n_2, \ldots\}\) mapping each arrival, as identified by its ordinal occurrence number \(i\), into its frame sequence number \(n_i\).
The arrival function assumes the natural order, that is, \( i < j \) implies \( n_i < n_j \).

A related function counts how many arrivals occur within a given interval:

**Definition 9.2.** Suppose \( \{s_n\} \) is a walk and \( Z \) is a set of steps. Let \( \varphi \) be an arrival function. The **counting function** \( N_Z : \mathbb{S}^N \times \mathbb{Z}^+ \to \mathbb{Z}^+ \) induced by \( \varphi \) is

\[
N_Z(\{s_n\}, k) = \max_{\varphi(i) \leq k} i,
\]

and for completeness set \( \max(\emptyset) = 0 \).

### 9.4. Normal operations.

The idea behind “normal operations” is a long program run following a software usage pattern. Interaction with the environment affects software behavior, which is ultimately transmitted through response to changing volatile variables. Normal operations calls for a run of figuratively unbounded duration during which software experiences the usage pattern’s variation of volatile stimulus, in response to which possibly unbalanced service is demanded from its inventory of functions.

Systems engineering often augments what is here the automaton’s step poset with a transition network of modes. These modes symbolically encapsulate enabled or disabled capabilities. However, even though this augmentation facilitates visualization of behaviors, it fails to be mathematically definitive.

#### 9.4.1. Orbit.

We have not defined normal operations, but every example would certainly constitute a walk (sequence of steps). A special walk illustrating normal operations (as described above) will be termed here an orbit. Without formal definition, use of this term sacrifices rigor.

**Remark.** The actuated automaton governs pure step transition logic, but an orbit also reflects a usage pattern.

#### 9.4.2. Limit conjecture.

Orbits may differ in specific sequence and content, but they have the same limit ratios. We consider a case drawn with the counting procedure of §9.3.

**Conjecture 9.3.** For different orbits \( o = \{s_n\} \) and \( o' = \{s'_n\} \) having the same usage pattern,

\[
\lim_{k \to \infty} \frac{N_{U'}(\{s_n\}, k)}{N_Z(\{s_n\}, k)} = \lim_{k \to \infty} \frac{N_{U'}(\{s'_n\}, k)}{N_Z(\{s'_n\}, k)}
\]

for sets of steps \( \emptyset \neq U \subseteq Z \subseteq \mathbb{S} \).

### 9.5. Types of operational profile.

A relative operational profile is the conditional probability that a step in an actuated automaton’s orbit coincides with a particular member of the reference set, given that it agrees with the reference set. We consider one other: an absolute operational profile is the time rate at which a particular step of an orbit coincides with any member of the reference set.

#### 9.6. Relative operational profile.

Let \( o = \{s_n\} \) be an orbit. Suppose \( z \in Z \subset \mathbb{S} \) is a step of the reference set. Software encounters \( N_{\{z\}}(\{s_n\}, k) \) instances of steps satisfying \( \{s_n\}(i) = s_i = z \) during the first \( k \) automaton steps. In the same execution there are \( N_Z(\{s_n\}, k) \) instances of \( s_i \in Z \). In the frequentist school of interpreting probability,

\[
P(z \mid Z) = \lim_{k \to \infty} \frac{N_{\{z\}}(\{s_n\}, k)}{N_Z(\{s_n\}, k)}
\]

represents the conditional probability of occurrence of \( z \), given that \( Z \) occurs. By Conjecture 9.3 every orbit (of the same usage pattern) yields the same relative operational profile.

A relative operational profile is an arbitrary set \( Z \) of steps, along with each step’s conditional probability of execution. In other words, a relative operational profile is a mapping \( O : Z \to [0, 1] \) having total measure 1.

---

5 The author is unaware of a proper definition.
9.7. **Absolute operational profile.** Let \( o = \{ s_n \} \) be an orbit. An absolute operational profile is the probability \( P(Z) \) with which an orbit (of some usage pattern) coincides with any step of the reference set \( Z \). As before, this probability is the limiting ratio of two counting functions. Its numerator contains \( N_{Z}(\{ s_n \}, k) \), the same count as appears in the denominator of the relative operational profile. In its denominator is the counting function of all possible steps, namely \( N_{S}(\{ s_n \}, k) \), where \( S \) is the space of all steps. Thus the ratio of counting functions of reference set \( Z \) to the entire space \( S \) is \( \frac{N_{Z}(\{ s_n \}, k)}{N_{S}(\{ s_n \}, k)} \), and the absolute operational profile (of collection \( Z \)) is

\[
P(Z) = \lim_{k \to \infty} \frac{N_{Z}(\{ s_n \}, k)}{k}.
\]

9.8. **Conversion into a rate.** Section 2.4.3 mentions the synchronization function, a cross-reference between discrete and real time. During each step, an amount of real time appropriate for a software system emulating the automaton’s step is added to the time consumption budget. Let \( Z \) be the usual arbitrary reference collection of steps and \( o = \{ s_n \} \) be an orbit. These two provide a set of events and a sequence of steps in which to count the events’ arrivals. The synchronization records discrete pairs \((i, t_i)\), where \( i \) is the index of the automaton step and \( t_k \) is the total elapsed time after \( k \) steps. Call this mapping the synchronization function, having the formalism \( \text{sync} : \mathbb{S}^N \times \mathbb{N} \to \mathbb{R}^+ \), along with assumed starting point \( \text{sync}(\{ s_n \}, 0) = 0 \).

Let the sequence index of each step be the discrete analog of time. Of course, this has the effect that discrete software time will not hold proportional to hardware real time. The approximate real time required by execution of step \( s = (\lambda, f, \mathbf{r}) \) is \( \tau(f) \) – that is, elapsed real time is taken as a function of the executing functionality.

**Definition 9.4.** For orbit \( o = \{ s_n \} \), approximate time elapsed during the first \( k \) steps accumulates to

\[
\text{sync}(\{ s_n \}, k) = \sum_{i=1}^{k} \tau(f_i) = \sum_{i=1}^{k} \tau(\mathcal{O}(s_n)) = t_k.
\]

A theorem to avoid creating dependency on specific orbits is in order. Inability to define normal operations leads instead to conjecture, expressing such need:

**Conjecture 9.5.** For different orbits \( o = \{ s_n \} \) and \( o' = \{ s'_n \} \) having the same usage pattern,

\[
\lim_{k \to \infty} \frac{N_{Z}(\{ s_n \}, k)}{\text{sync}(\{ s_n \}, k)} = \lim_{k \to \infty} \frac{N_{Z}(\{ s'_n \}, k)}{\text{sync}(\{ s'_n \}, k)}
\]

for a reference set of steps \( \emptyset \neq Z \subseteq S \).

The synchronization function allows expression of the absolute operational profile as an intensity (rate or quasi-frequency).

**Definition 9.6.** The counting norm is written using the double bar notation \( \| \cdot \| \):

\[
\| Z \| = \lim_{k \to \infty} \frac{N_{Z}(\{ s_n \}, k)}{\text{sync}(\{ s_n \}, k)}.
\]

The absolute operational profile is properly a subadditive seminorm on sets of steps. As the limiting ratio of two counts in the natural numbers, the norm is positive. The norm is a seminorm because for some nonempty set \( Z \) it may be true that \( \| Z \| = 0 \) (if the usage pattern does not activate any member of the reference set). This norm is subadditive because for any other set \( S \), \( N_{Z \cup S}(\{ s_n \}, k) \leq N_{Z}(\{ s_n \}, k) + N_{S}(\{ s_n \}, k) \). It follows that \( \| Z \cup S \| \leq \| Z \| + \| S \| \).
10. Software safety (reliability) demonstration

A reliability demonstration is any structured experiment carrying controlled statistical uncertainty and providing "hard" evidence against potential liability. In safety analysis, the demonstration’s focus is a specific hazard in a particular product. We especially desire to formulate a random experiment to indemnify that a transduction from software to hardware is safe. One such random experiment is a safety demonstration.

The principle of emergence (§2.5) posits that software causes no harm until erroneous values transduce the boundary between software and hardware. The cone mechanism is appreciated when its crux step coincides with a point of transduction.

10.1. Safety demonstration. In oversimplification, a safety demonstration is a random sample from an acyclic cone. The complete story is not so simple, because the cone is not a probabilistic structure; it possesses no probability to support randomness.

As a probabilistic structure, the operational profile (§9.6) permits random sampling from its reference set, regardless of its higher level meaning. As the edge of a cone is a set, it can become a relative operational profile’s reference set. Thus we tie an operational profile to a cone’s edge: let \( O : \text{edge} C \rightarrow [0, 1] \) be a relative operational profile on the edge of cone \( C \). At this stage we have the ability to draw a random sample from edge \( C \).

Theorem 8.11 asserts that an acyclic cone \( C \) and edge \( C \) are in one-to-one correspondence via the edge step relation of a localized predecessor walk. Equivalent to the one-to-one correspondence is the bijection \( b = \{(\text{edge} w, w) : w \in C \} \). For \( e \in \text{edge} C \), \( b(e) \) is the bijectively corresponding localized predecessor walk.

We now bijectively associate the random edge event \( e = b^{-1}(w) \) with the localized predecessor walk \( w : O' = \{(O(b^{-1}(w)), w) : w \in C \} \). With probability inherited from an operational profile, we can speak validly of a random sample from a cone.

10.2. Tests. The last piece of the safety demonstration story is converting local predecessor walks into tests. Localized predecessor walks are finite walks existing in confusion-prone backwards time. The test function reverses and re-indexes localized predecessor walks into conventional sequences.

Definition 10.1. Let \( w \) be a localized predecessor walk of \( n = |w| \) steps, indexed from 0 down to \(-(n - 1)\). Define the test function \( t(w) = \tilde{w} \) according to formula \( \tilde{w}_i = w_{i-n} \) for \( i = 1, 2, \cdots, n \).

Assuming that a localized predecessor walk is indexed from 0 down to \(-(n - 1)\), its corresponding test will be indexed from 1 to \( n \). In sense of direction, the localized predecessor walk traverses steps from \( s_{\text{exu}} \) to \( s_{\text{edge}} \), while the corresponding test traverses steps from \( s_{\text{edge}} \) to \( s_{\text{exu}} \).

Theorem 10.2. Suppose \( C \) is an acyclic cone and \( \mathbb{W} \subset C \) is a (unique) set of localized predecessor walks. If \( \mathbb{W} = t(\mathbb{W}) \) is its converted set of reversed and re-indexed tests, then \( \mathbb{W} \) and \( \mathbb{W} \) are in one-to-one correspondence.

Proof. By virtue of construction, \( t \) is already a mapping. Remaining to show is that \( t \) is additionally a bijection. Let \( u \) and \( v \) be localized predecessor walks and \( x \) be a finite walk. As hypothesis set \( x = t(u) = t(v) \). These sequences cannot be equal unless they possess the same number of terms, \( n = |x| = |t(u)| = |t(v)| \). Since transformation \( t \) preserves the number of steps (from Definition 10.1 \( |w| = |t(w)| \)), then \( n = |x| = |u| = |v| \).

Again invoking Definition 10.1 on the first part of the hypothesis, we write \( x_i = u_{i-n} \). The second part similarly yields \( x_i = v_{i-n} \). By equating the two parts, we now have \( u_{i-n} = v_{i-n} \) for each \( i \). In other words, the two localized predecessor walks are actually the same walk: \( u = v \). Thus \( t \) is a bijection. \( \square \)
10.2.1. **Volatile variables preservation.** The danger in reversed thinking about tests is inadvertently conceptualizing volatile variables as free. This is untrue, as the volatile variables at any stage of a predecessor chain are fixed, and the “next” stage considers the set of what previous conditions may have led to the current stage. Thus, predecessor walks are chains of a poset of steps, which include the settings of volatile variables. One must be mindful to reproduce all volatile stimuli of the localized predecessor walk in its analogous test.

10.3. **Outcome.** The outcome of a test, pass or fail, will be regarded as a Bernoulli random event, \( P_\rho = \rho^n (1 - \rho)^{1-n} \), for \( n = 1 \) (pass) or \( n = 0 \) (fail). These probabilities are statistically independent of the bias involved with drawing the sample from the operational profile. This bias affects the origin of discovered failures, not how many failures are found. In other words, the total statistical power of the sampling plan is not affected by sampling bias.

Sum of independent Bernoulli random variables are binomial. That is, the probability of finding \( n \) failures collectively among \( N \) sample items is binomial, \( \binom{N}{n} \rho^n (1 - \rho)^{N-n} \).

10.4. **Physics.** In the real world, tests pass or fail depending on whether the information transduced at step \( s_{crux} \) meets all safety constraints. Such engineering requirements are varied, ultimately involving position, timing, voltage, insulation, dimensional tolerance, toxicity, temperature, mechanical shielding, luminosity, and hydrostatic pressure – just to name a few areas. Review of a test offers a last chance to discover a missed constraint (requirement). Another possibility is that the chain of precursor events should actually lead to a different conclusion.

Transduced values potentially control the status of any safety concern. Tests simply pass or fail, but evaluation of why a test passes or fails can become nontrivial, requiring collaboration between mechanical, software, and system safety engineers.

10.5. **Statistics.** Some statistical error originates in inference from random sample to “unknown” population (parametric family of probability distributions on a measurable space). Just one distribution is true, while the others are false. An assertion separating the parameterization into two decision units is called a hypothesis. One decision unit is traditionally designated null, while the other is called alternate. The true distribution belongs either to the null or alternative decision units.

Each sample item either passes or fails its associated test (see §10.4). Within the entire cone \( C \), suppose the proportion of tests that fail is \( \rho \). This proportion is subsequently realized approximately through a random sample. Regardless of the sample size, since the application is to safety, the only cases of interest will be when the number of failures is zero. Other cases, implying need for reliability growth, are treated in the literature, particularly [3].

We now examine the case defined by drawing a random sample of size \( N \) from edge \( C \) and allowing \( n = 0 \) failures in the associated tests from cone \( C \). The null decision unit contains the probability distribution \( P_0(\text{pass}) = 1 \) and \( P_0(\text{fail}) = 0 \). The alternate decision unit is the set of probability distributions \( P_\rho \) having \( 0 < \rho \leq 1 \). Hypothesis evaluation entails two types of error, known as \( \alpha \) and \( \beta \) error.

10.5.1. **False rejection (\( \alpha \) error).** The first is false rejection of the null decision unit, with associated measurement error \( \alpha \). The sampling plan can reject only if finds an error, so this sampling plan is incapable of false rejection. Thus \( \alpha \equiv 0 \).

10.5.2. **False acceptance (\( \beta \) error).** The second is false acceptance of the null decision unit, with associated measurement error \( \beta \). We experience false acceptance when \( 0 < \rho \) but the sample contains no failures.

Under the binomial model, the probability of observing a random sample of size \( N \) with \( n \) failures collectively is \( \binom{N}{n} \rho^n (1 - \rho)^{N-n} \). Proceeding to our case of interest, \( n = 0 \), we have \( \binom{N}{0} \rho^0 (1 - \rho)^{N-0} \mid_{n=0} = \binom{N}{0} (1 - \rho)^N \).
\[(1 - \rho)^N.\] This expression is the probability that random samples of size \(N\) from a source of characteristic failure proportion \(\rho\) will be accepted.

10.5.3. **Power function.** It is confusing to reason in terms of contravariant attributes. In our case we formulate probability of rejection as an increasing function of \(\rho\), a measure of the population’s undesirability. The probability that random samples will be properly rejected is the previous expression’s complement:

\[K_{N,0}(\rho) = 1 - (1 - \rho)^N = 1 - \beta.\]

This non-contravariant result is known as the power function of sample size \(N\), tolerating zero \((0)\) failures. The graph of the power function always increases, starting at 0 for \(\rho = 0\) and ending at 1 for \(\rho = 1\). Just how fast this function increases in its midrange is determined by the sample size \(N\). With sample size one \((N = 1)\), \(K_{1,0}(\rho) = \rho\).

| \(N\) | \(K_{N,0}(0.001)\) | \(K_{N,0}(0.01)\) | \(K_{N,0}(0.05)\) | \(K_{N,0}(0.10)\) | \(K_{N,0}(0.50)\) | \(K_{N,0}(0.90)\) |
|------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 1    | .0010           | .0100           | .0500           | .1000           | .5000           | .9000           |
| 5    | .0050           | .0490           | .2626           | .4095           | .9688           | 1.0000          |
| 10   | .0100           | .0956           | .4013           | .6513           | .9990           | 1.0000          |
| 15   | .0149           | .1399           | .5367           | .7941           | 1.0000          | 1.0000          |
| 20   | .0198           | .1821           | .6415           | .8784           | 1.0000          | 1.0000          |
| 30   | .0296           | .2603           | .7854           | .9576           | 1.0000          | 1.0000          |
| 50   | .0488           | .3950           | .9231           | .9948           | 1.0000          | 1.0000          |
| 100  | .0952           | .6340           | .9941           | 1.0000          | 1.0000          | 1.0000          |
| 200  | .1814           | .8660           | 1.0000          | 1.0000          | 1.0000          | 1.0000          |
| 500  | .3936           | .9934           | 1.0000          | 1.0000          | 1.0000          | 1.0000          |
| 1000 | .6323           | 1.0000          | 1.0000          | 1.0000          | 1.0000          | 1.0000          |
| 2000 | .8648           | 1.0000          | 1.0000          | 1.0000          | 1.0000          | 1.0000          |
| 5000 | .9933           | 1.0000          | 1.0000          | 1.0000          | 1.0000          | 1.0000          |
| 10000| 1.0000          | 1.0000          | 1.0000          | 1.0000          | 1.0000          | 1.0000          |

TABLE 1. Family of power functions (probability of rejection)

Within this family \(\beta = (1 - \rho)^N = 1 - K_{N,0}(\rho)\).

One is initially dismayed by this sketch of the family of power functions; it suggests that high degrees of assurance are unobtainable through random sampling using practical sample sizes. However, reasonable performance useful for coarser screening is very possible. Detecting a defective population of 10 percent with a probability of approximately 90% requires only 20 sample items.

10.6. **Sampling philosophy.** Our safety demonstration sampling technique contrasts two assurance philosophies – software reliability versus software correctness. The software reliability perspective involves a separate operational profile on edge \(C\), whereas software correctness examines only the structure within cone \(C\). The operational profile asserts the importance of relative excitational intensity to safety analysis. An accident that occurs more frequently is worse than an accident that happens less frequently, given that they are of comparable severity. This safety factor is ignored under software correctness alone.

\(\text{？One increasing, the other decreasing} \)
11. Modeling accidents

Accidents are diverse in effect and mechanism, including injury, death, or damage either to equipment or environment. Since the causality of accidents is temporarily unknown, they manifest an apparent nature of unpredictability or randomness. However, under emulation as a stochastic process, the exact timing of accidents is truly a random phenomenon rather than causal. Nevertheless, it has proven useful to compare well-understood summary statistics of stochastic processes with those of deterministic but unknown physical processes.

11.1. Compound Poisson process. Today’s prevalent safety model for the occurrence of accidents is the compound Poisson process. This model captures accidents’ two dominant attributes: rate of occurrence (intensity) and scalar measure of loss (severity). With some exceptions, neither the timing nor severity of one software accident affects another. The compound Poisson process (CPP) is appropriate to model accidents of this nature.

As stochastic processes are models rather than mechanisms, deriving their properties involves somewhat out-of-scope mathematics. The interested reader can immediately find greater detail in Wikipedia online articles: [11], [12], [13], [14], and [15]. Relevant theorems will be documented here simply as facts.

11.2. Poisson processes. We will consider three variants of basic stochastic process: the ordinary Poisson process, the compound Poisson process, and the intermittent compound Poisson process.

11.2.1. Ordinary Poisson process. (Ordinary) Poisson processes are characterized simply by their rate or intensity:

- its fundamental rate \( \lambda \), which is the expected number of arrivals per unit time.

**Fact 11.1.** Let \( \lambda \) be the rate of a Poisson process. The probability of experiencing \( k \) arrivals in a time interval \( t \) units long is

\[
P_\lambda(k) = e^{-\lambda t} \frac{\lambda^k}{k!}.
\]

11.2.2. Compound Poisson process. A compound Poisson process is characterized by two rates:

- its fundamental rate \( \lambda \) as before, and
- its rate of loss \( L \), which is a random variable invoked once for each arrival.

**Fact 11.2.** Let \( \lambda \) be the rate and \( L \) be the loss random variable of a compound Poisson process. The expectation of the compound process for a time interval \( t \) units long is

\[
E(\text{compound Poisson}) = \lambda t \cdot E(L)
= \lambda t \cdot \mu_L.
\]

**Definition 11.3.** The statistical risk, written \( h \), of a compound Poisson process is the time derivative of its expectation in a duration of length \( t \); that is

\[
h = \frac{d}{dt} E(\text{compound Poisson}) = \frac{d}{dt} (\lambda t \cdot \mu_L) = \lambda \mu_L,
\]

which is the product of its rate \( \lambda \) and its expected loss \( \mu_L \).

---

7 After Siméon Denis Poisson, mathematician and physicist, 1781 – 1840
11.2.3. **Intermittent compound Poisson process.** A variation of the CPP is the intermittent compound Poisson process, which is intermittently on or off with expected durations $E(on) = \mu_{on}$ and $E(off) = \mu_{off}$. An intermittent compound Poisson process (ICPP) is characterized by three rates:

- its fundamental rate $\lambda$ as before, and
- its rate of loss $L$, also as before,
- alternating durations of random lengths $\tau_{on}$ and $\tau_{off}$.

Random variables $\tau_{on}$ and $\tau_{off}$ converge to $\mu_{on}$ and $\mu_{off}$ in the limit. The *idle* ratio of an intermittent compound Poisson process is $\iota = \frac{\mu_{off}}{\mu_{on} + \mu_{off}}$.

**Fact 11.4.** Let $\lambda$ be the rate, $L$ be the loss random variable, and $\iota$ be the idle ratio of an intermittent compound Poisson process. The expectation of the ICPP for a time interval $t$ units long is

$$E(\text{intermittent compound Poisson}) = (1 - \iota) \cdot \lambda t \cdot E(L) = (1 - \iota) \cdot \lambda t \cdot \mu_L.$$ 

The statistical risk of an ICPP is

$$h = \frac{d}{dt} E(\text{intermittent compound Poisson})$$

$$= \frac{d}{dt} ((1 - \iota) \cdot \lambda t \cdot \mu_L + \iota \cdot 0 t \cdot 0)$$

$$= (1 - \iota) \lambda \mu_L.$$ 

12. **INDEMNIFICATION**

Hypothesize that a software hazard is emulated by a compound Poisson process (CPP) having intensity $\lambda$ and expected loss $\mu_L$. Suppose further that the actual control mechanism is a cone convergent to the software point of exhibition of the hazard. We wish to consider statistical evidence that the hazard’s hypothetical description via the stochastic process is consistent with its mechanism as revealed by safety demonstration.

12.1. **Unification.** Before undertaking the question of whether test data supports a hypothetical stochastic process, we must establish the theoretical conditions under which equality is expected.

12.1.1. **Fundaments of the model.** The compound Poisson process is a model stochastic process for occurrence of accidents. This model is used in safety analysis to quantify the occurrence and losses of accidents without considering their causes. MIL-STD-882 (see Appendix C) is an important example. In a time interval of duration $t$, accidents converge stochastically in rate to expectation $\lambda t$ and in mean loss to $\mu_L$. This means an intensity of $\lambda$ accidents per time unit.

12.1.2. **Fundaments of the mechanism.** The actuated automaton is a mechanism representing software. When extended by the principle of emergence (§2.5) and the constructs of the operational profiles (§9) and cones (§8), it becomes capable of representing precursor conditions for software accidents. Let $\|\text{edge} \ C\|$ (see §9.8) be the rate-based absolute operational profile of the edge of an acyclic cone $C$. Since a member of $\text{edge} \ C$ is executed at the average intensity of $\|\text{edge} \ C\|$, then so is the cone’s step of convergence $s_{\text{crux}}$. Let $\rho$ be the proportion of failing tests (localized predecessor walks). Under that supposition, failures occur at the intensity of $\rho \cdot \|\text{edge} \ C\|$. The definition of $\|\text{edge} \ C\|$, through the internal function $\text{sync}(\cdot)$, allows for the passage of time in the proper duration.
12.1.3. **Uniting mechanism and model.** We presume that one failing test equals one accident. The cone’s step of convergence is considered to be the point of exhibition of a hazard whenever safety constraints are not met. This mechanism may be separately equated to the intensity (not the rate of loss) of the compound Poisson process:

\[ \lambda = \rho \cdot \| \text{edge} C \|. \]

This equation places a property of the model on the left and properties of the mechanism on the right.

**Remark.** The execution rate of the edge of an acyclic cone numerically equals the execution rate of the (set containing the) cone’s crux. The cone’s definitional status (as a complete independent set of localized predecessor walks ending at \( s_{\text{crux}} \)) causes this. Symbolically,

\[ \| \text{edge} C \| = \| \{ s_{\text{crux}} \} \|. \]

12.2. **Evidence.** We propose that the same data used earlier for indemnification testing be re-used in a slightly different statistical context. Recall that an indemnification test has \( N \) items among which are zero failures, where each item is a localized predecessor walk, and a cone is a structured collection of localized predecessor walks of an automaton.

We wish to measure the amount of information in an indemnification sample to explain the phenomenon that larger samples justify more precise estimates than smaller samples. We refer to this information as measuring the *weight of evidence*. This situation differs from the familiar problem of finding the maximum likelihood estimator.

12.2.1. **Method of indifference.** The power function of sample size \( N \), tolerating zero (0) failures, is \( K_{N,0}(\rho) = 1 - (1 - \rho)^N \) (see §10.5.3). It measures the probability of rejection as a function of \( \rho \).

Each power function \( K_{N,0}(\rho) = 1 - (1 - \rho)^N \) is characterized by its indifference proportion, which is defined as the proportion at which rejection and acceptance become equally likely (that is, \( K_{N,0}(\hat{\rho}_1) = \frac{1}{2} = 1 - K_{N,0}(\hat{\rho}_1) \)). With only modest algebra, the analytic expression for the indifference proportion may be derived from the power function \( K_{N,0}(\rho) \); it is

\[ \hat{\rho}_1 = 1 - \sqrt[2]{\frac{1}{2}}. \]
Below is a numerical tabulation of the previous formula:

| \( N \) | \( \hat{\rho}_{\text{indifference}} \) |
|---|---|
| 1  | .50000 |
| 5  | .12945 |
| 10 | .06697 |
| 15 | .04516 |
| 20 | .03406 |
| 30 | .02284 |
| 50 | .01377 |
| 100| .00691 |
| 200| .00346 |
| 500| .00139 |
| 1000| .00069 |
| 2000| .00035 |
| 5000| .00014 |
| 10000| .00007 |

Table 2. Indifference proportion

12.2.2. **Indemnification formula.** The indemnification formula provides a statistical upper bound on hazard intensity. Indemnification data may be expressed as an equivalent statistical upper bound on hazard intensity. This differs fundamentally from estimating the intensity of a hazard. By a statistically “guaranteed” hazard intensity, we mean an upper bound such that the true hazard intensity is likely to fall beneath this level with known confidence (probability).

Suppose we choose \( \frac{1}{2} \) as the known confidence. The indifference proportion \( \hat{\rho}_I = 1 - \sqrt{\frac{1}{2}} \) then has a second interpretation as an upper bound with confidence \( \frac{1}{2} \). For any \( \rho \leq \hat{\rho}_I \), it is true that power function \( P_{N,0}(\rho) \leq \frac{1}{2} \), so \( \hat{\rho}_I \) is an upper bound of confidence \( \frac{1}{2} \).

To convert from the size of the indemnification sample into its equivalent upper bound hazard intensity, find the indifference proportion \( \hat{\rho}_I = 1 - \sqrt{\frac{1}{2}} \).

Check the sample physics. This amounts to analysis of the originating cone \( \mathcal{C} \), which is the point of exhibition of a hazard whenever safety constraints are not met. The cone’s edge has an absolute operational profile expressed as a rate. This quantity is the counting norm of the cone’s edge.

We have shown that the probable upper bound of the hazard intensity is proportional to the indifference proportion, with constant of proportionality furnished by the counting norm of the cone’s edge. The indemnification formula is:

\[
\hat{\lambda}_I = \hat{\rho}_I \cdot ||\text{edge}\mathcal{C}||.
\]
13. **Epilogue**

From this mathematical discussion have emerged two unambiguous structures of system safety: the safety demonstration and indemnification, its measure of assurance. Implication and opinion follows.

13.1. **Harnessing nascent science.** Safety demonstration and indemnification merge smoothly into today’s programmatic picture. Early in the development cycle, safety engineers provide “ballpark” quantifications of the threat of hazards[7], expressed as intensity and severity. These numbers are often educated guesses: a mixture of circumstance, intuition, similar design, and history. At that stage, the process is without supporting evidence. Later in the development cycle, assuming a program of structured testing has been followed, statistical evidence is available in the form of random samples from the execution of a cone – that is, a safety demonstration. These data are expressed as a statistical upper bound on each software hazard’s intensity – that is, an indemnification – and used as evidence of correct operation. As has been shown, increased sample sizes result in greater statistical confidence, or numerically lower indemnifications.

This evidence would correct an existing imbalance in software assurance, favoring sub-process completion or other events which a software quality team could monitor. Increasingly, automatic static syntax analyzers satisfy need for general code robustness. Responsibility for a program of safety demonstration and indemnification would create a unifying focus for the safety engineering discipline. A program of indemnification is based on specific hazards in a particular product, and is arguably an important factor in safety management.

13.2. **Commercialization.** Difficult work remains before safety demonstration and indemnification can be supported as mature commercial technology. The role of the actuated automaton must be replaced by a real-world programming language. Present theory restricting tests (predecessor walks) to acyclic cones may require generalization to cyclic cones to achieve broader range. Safety demonstration demands the ability to produce approximate operational profiles from which can be drawn pseudo-random samples. Spin-offs from similar technologies may be possible; static analyzers are one example.

13.3. **Criticism.** This mathematical discussion concludes that the management of software hazards under JSSSEH[6] and MIL-STD-882[7] need not deviate philosophically from hardware practice. Versioning the meaning of the term ‘hazard’ has not been justified as necessary, as neither has modification of the system of evaluating hazards by frequency and severity. Indeed, unification is possible – this essay presents one alternative yielding homogeneous hazard philosophy. The versioned systems of reference lack a recommendation for ranking the importance of hazards of mixed type (hardware and software). This outcome is symptomatic of these documents’ splintered philosophies. At heart, such outcomes are antithetical to the spirit of systems.

13.4. **Recommendation.** Responsible parties should evolve away from approaches to system safety that partition the discipline unnecessarily into branches. Such approaches should be tolerated as temporary stopgaps serving only the empirical needs of the moment. Improved levels of integration must be continually pursued, one candidate among which is the subject of this essay. Bearing in mind that improved methods may entail new forms of technology support, software tool vendors must keep abreast of emerging needs. To facilitate awareness, maintainers of standards could sponsor web-hosted conferences, in which professional societies could maintain a presence.
APPENDIX A. GROUNDWORK

This section examines ensembles, a fundamental structure in the theory of systems which, through the Cartesian product and its resulting choice space, underpins the concepts of stimulus, response, state and event.

A.1. Ensemble. An ensemble is a special form of a more general structure known as a family. We employ nomenclature abridged from Halmos [5, p. 34] as follows:

**Terminology.** Let \( I \) and \( X \) be non-empty sets, and \( \varphi : I \rightarrow X \) be a mapping. Each element \( i \in I \) is an index, while \( I \) itself is an index set. The mapping \( \varphi \) is a family; its co-domain \( X \) is an indexed set. An ordered pair \((i, x)\) belonging to the family is a term, whose value \( x = \varphi(i) \in X \) is often denoted \( \varphi_i \).

**Notation.** The family \( \varphi \) itself is routinely but abusively denoted \( \{ \varphi_n \} \). This notation is a compound idiom. Especially in the case of sequences over a set \( G \), the symbol \( \{ g_n \} \) signifies the mapping \( \{ 1 \mapsto g_1, 2 \mapsto g_2, \ldots \} \).

**Definition A.1.** An ensemble is a non-empty family.

**Definition A.2.** Let \( \Psi \) be an ensemble. If \( |\Psi| = 1 \), the ensemble is constant; otherwise it is variable.

**Remark.** Since physical systems possess only a finite number of attributes, the scope of practical interest is limited to ensembles having finite index sets.

**Remark.** A constant ensemble (definition A.2) is also referenced under the historically colorful name Hobson’s choice. Using the word choice in its everyday sense, a Hobson’s choice is oxymoronic: a free choice in which only one option is offered, with gist “Only one choice is no choice.”

**Notation.** Let \( \Psi \) be an ensemble. For term \((i, P) \in \Psi\), we denote \( P = \Psi_i \).

A.2. Ensemble arithmetic.

**Definition A.3.** Ensembles \( \Psi \) and \( \Phi \) are disjoint if \( \text{dom } \Psi \cap \text{dom } \Phi = \emptyset \) (that is, if their index sets have no member in common).

**Definition A.4.** Ensembles \( \Psi \) and \( \Phi \) are complementary with respect to a third ensemble \( \Upsilon \) if they are disjoint and \( \Psi \cup \Phi = \Upsilon \).

**Notation.** Regarding ensembles \( \Psi \) and \( \Phi \), we write \( \Phi \subseteq \Psi \) to express that \( \Phi \) is contained in \( \Psi \), following ordinary set theory that term \((i, P) \in \Phi \) implies \((i, P) \in \Psi \).

**Definition A.5.** Let \( \Psi \) and \( \Phi \) be ensembles such that \( \Phi \subseteq \Psi \). In classification of difference between \( \Psi \) and \( \Phi \), \( \Psi \) is the minuend, \( \Phi \) is the subtrahend, and the set difference \( \Psi \setminus \Phi \) is the remainder.

**Lemma A.6.** Let \( \Psi \) and \( \Phi \) be ensembles such that \( \Phi \subseteq \Psi \). The subtrahend \( \Phi \) and remainder \( \Psi \setminus \Phi \) are disjoint and complementary with respect to \( \Psi \).

**Proof.** Since \( \Phi \subseteq \Psi \), definition A.5 applies. The minuend is \( \Psi \), the subtrahend is \( \Phi \), and the remainder is \( \Psi \setminus \Phi \).

Definition A.3 asserts that two ensembles \( \Theta \) and \( \Upsilon \) are disjoint if \( \text{dom } \Theta \cap \text{dom } \Upsilon = \emptyset \). As hypothesis presume the lemma’s antithesis, namely that the subtrahend \( \Phi \) and the remainder \( \Psi \setminus \Phi \) are not disjoint. This implies that there exists some \( i \) such that \( i \in \text{dom } \Phi \cap \text{dom } (\Psi \setminus \Phi) \). For this \( i \) to exist in the intersection of the domains of two mappings, there must be both a term \((i, P) \in \Phi \) and a term \((i, Q) \in (\Psi \setminus \Phi) \).

---

8 Its symbol is a composite of other notational devices that have no separate meaning.

9 Hobson was proprietor of a livery. He was noted for offering his customers their choice of any horse, as long as that horse was in the first unoccupied stall.

10 The set difference \( A \setminus B \) is not conventionally restricted to \( B \subseteq A \), as is stipulated here.
Because \((i, P) \in \Phi\) and the lemma’s premise states that \(\Phi \subseteq \Psi\), then \((i, P) \in \Psi\). Since term \((i, Q) \in (\Psi \setminus \Phi)\), then \((i, Q) \in \Psi\) and \((i, Q) \notin \Phi\). Since \(\Psi\) is a mapping, then \((i, P) \in \Psi\) and \((i, Q) \in \Psi\) together imply \(P = Q\). With \(P = Q\) and \((i, Q) \notin \Phi\), then also \((i, P) \notin \Phi\).

However, the immediately preceding conclusion that \((i, P) \notin \Phi\) contradicts the earlier inference that term \((i, P) \in \Phi\) must exist if \(i\) is a member of the intersection of the domains. Since the presumption that the subtrahend and remainder are not disjoint arrives at a contradiction, then subtrahend \(\Phi\) and remainder \(\Psi \setminus \Phi\) are indeed disjoint.

The ensembles are complementary with respect to \(\Psi\) by definition \ref{def:complementary}, because they are disjoint and \(\Phi \cup (\Psi \setminus \Phi) = \Psi\). \qed

### A.3. Choice space

Informally, a choice space is the totality of all possible combinations of variables’ values within a given ensemble. The general Cartesian product formalizes this notion.

#### A.3.1. General Cartesian product

**Definition A.7.** Let \(\Psi\) be an ensemble. By definition \ref{def:proto-set}, each member of \(\text{ran} \ \Psi\) is itself a non-empty set. The proto-set \(\Psi\circ\) is the union of all such sets:

\[
\Psi\circ = \bigcup_{R \in \text{ran} \ \Psi} R.
\]

**Remark.** Definition \ref{def:proto-set} expresses a relation between the ensemble \(\Psi\)’s indexed sets and the proto-set, namely that for each \(i \in \text{dom} \ \Psi\), \(\Psi(i) \subseteq \Psi\circ\). An alternative portrayal uses the power set, claiming \(\Psi(i) \in \mathcal{P}(\Psi\circ)\).

**Definition A.8.** Let \(\Psi\) be an ensemble with proto-set \(\Psi\circ\). The proto-space of \(\Psi\) is the set \(\Psi\circ_{\text{dom} \ \Psi}\), where \(\Psi\circ_{\text{dom} \ \Psi}\) is the set of all mappings \(\text{dom} \ \Psi \to \Psi\circ\).

**Definition A.9.** Let \(\Psi\) be an ensemble with proto-set \(\Psi\circ\). Let \(\chi\colon \text{dom} \ \Psi \to \Psi\circ\) be a mapping. If \(\chi\) satisfies \(\chi(i) \in \Psi(i)\) for each \(i \in \text{dom} \ \Psi\), then \(\chi\) is a choice mapping of \(\Psi\).

**Definition A.10.** The set of all choice mappings of ensemble \(\Psi\) is the choice space (or general Cartesian product) \(\prod \ \Psi\).

**Terminology.** Through the general Cartesian product, an ensemble generates a choice space. For brevity we refer to a point in a choice space (that is, a choice mapping) simply as a choice.

**Theorem A.11.** The proto-space \(\Psi\circ_{\text{dom} \ \Psi}\) of ensemble \(\Psi\) includes its choice space \(\prod \ \Psi\) (that is, \(\prod \Psi \subseteq \Psi\circ_{\text{dom} \ \Psi}\)).

**Proof.** Suppose \(\chi \in \prod \Psi\). By definition \ref{def:choice-space}, \(\chi\) is a mapping \(\text{dom} \ \Psi \to \Psi\circ\). Then, by definition \ref{def:proto-space}, \(\chi \in \Psi\circ_{\text{dom} \ \Psi}\). Thus, any member of \(\prod \Psi\) is also a member of \(\Psi\circ_{\text{dom} \ \Psi}\). From this we conclude \(\prod \Psi \subseteq \Psi\circ_{\text{dom} \ \Psi}\). \qed

**Theorem A.12.** An ensemble generates one unique choice space: let \(\Theta\) and \(\Phi\) be two ensembles generating choice spaces \(\prod \Theta\) and \(\prod \Phi\) respectively. If \(\Theta = \Phi\), then \(\prod \Theta = \prod \Phi\).

**Proof.** To show the contrapositive, suppose \(\prod \Theta \neq \prod \Phi\). This premise can be true if either A: there is a choice \(\alpha \in \prod \Theta\) such that \(\alpha \notin \prod \Phi\), or if B: there is a choice \(\beta \in \prod \Phi\) such that \(\beta \notin \prod \Theta\).

In case A, \(\alpha\) is a choice of \(\Theta\) – that is, for each \(k \in \text{dom} \ \Theta\), \(\alpha(k) \in \Theta(k)\). For hypothesis, assume \(\Theta = \Phi\), so that \(\text{dom} \ \Theta = \text{dom} \ \Theta\). Then for each \(k \in \text{dom} \ \Theta\), \(\alpha(k) \in \Phi(k)\), since by equality hypothesis \(\Theta(k) = \Phi(k)\) and \(\alpha(k) \in \Theta(k)\). This means \(\alpha\) is a choice of \(\Phi\), that is, \(\alpha \in \prod \Phi\). However, this conclusion contradicts the second part of the premises for case A, namely that \(\alpha \notin \prod \Phi\). Thus the hypothesis \(\Theta = \Phi\) is false, so \(\Theta \neq \Phi\).
The argument for case B is the same as A, except reversing the roles of $\Theta$ and $\Phi$. Together, cases A and B show that $\prod \Theta \neq \prod \Phi$ implies $\Theta \neq \Phi$. Applying the contrapositive principle\footnote{one form of which is $(\neg B \Rightarrow \neg A) \Leftrightarrow (A \Rightarrow B)$} gives $\prod \Theta = \prod \Phi$ if $\Theta = \Phi$.

**Theorem A.13.** Any choice space has one unique generating ensemble: let $\Theta$ and $\Phi$ be two ensembles generating choice spaces $\prod \Theta$ and $\prod \Phi$ respectively. If $\prod \Theta = \prod \Phi$, then $\Theta = \Phi$.

**Proof.** To show the contrapositive, suppose $\Theta \neq \Phi$. From this premise there must exist either A: $(i, P) \in \Theta$ such that $(i, P) \notin \Phi$, or B: $(j, Q) \in \Phi$ such that $(j, Q) \notin \Theta$.

The first case A decomposes into two sub-cases: either A1: $i \in S$ and $i \notin S'$, or A2: $i \in S$ and $i \notin S'$.

In sub-case A1, we must have $\Theta(i) \neq \Phi(i)$ to support $\Theta \neq \Phi$. For this there must be either A1a: there is $u \in \Theta(i)$ such that $u \notin \Phi(i)$, or A1b: there is $v \in \Phi(i)$ such that $v \notin \Theta(i)$.

In sub-sub-case A1a, there must exist by definitions A.10 and A.9 a choice $\alpha \in \prod \Theta$ such that $\alpha(i) = u$. However, there can be no choice $\beta \in \prod \Phi$ such that $\beta(i) = u$, since $u \notin \Phi(i)$. Therefore $\alpha \notin \prod \Phi$, with the consequence that $\prod \Theta \neq \prod \Phi$.

In sub-sub-case A1b there is $v \in \Phi(i)$ such that $v \notin \Theta(i)$. Similar to A1a, there exists $\beta \in \prod \Phi$ such that $\beta(i) = v$, but no $\alpha \in \prod \Theta$ such that $\alpha(i) = v$, again concluding that $\prod \Theta \neq \prod \Phi$.

In sub-case A2, we have $i \in \text{dom} \ \Theta$ and $i \notin \text{dom} \ \Phi$. By definitions A.10 and A.9 for any $x \in \Theta(i)$, there exists choice $\alpha_{x} \in \prod \Theta$ such that $\alpha_{x}(i) = x$. However, there is no choice $\beta \in \prod \Phi$ such that $\beta(i) = x$ because $i \notin \text{dom} \ \Phi = \text{dom} \ \beta$. This concludes $\prod \Theta \neq \prod \Phi$ for sub-case A2.

The proofs for subordinate cases of B are the same as A, reversing the roles of $\Theta$ and $\Phi$. Taken together, these sub-case analyses show that exhaustively $\Theta \neq \Phi$ implies $\prod \Theta \neq \prod \Phi$. By contraposition, $\Theta = \Phi$ if $\prod \Theta = \prod \Phi$.

**Theorem A.14.** Choice spaces and generating ensembles are in one-to-one correspondence via the general Cartesian product.

**Proof.** Theorem A.12 asserts that any ensemble generates only one unique choice space, and A.13 asserts that any choice space has only one unique generating ensemble.

**Notation.** By Theorem A.14 the Cartesian product is invertible, providing a mapping from choice spaces to generating ensembles. There is need for a symbol designating this inverse. In preference to $\prod^{-1}$, for present purpose we borrow the coproduct symbol $\coprod$ from another field, since there is no danger of confusion.

**Definition A.15.** A finite choice space is a choice space (definition A.10) whose generating ensemble has a finite index set.

A.3.2. Dyadic Operators.

**Notation.** This section presents binary operators on ensembles and choice spaces whose intended usage will value notational compactness. To this end these operators will be displayed through dyadic notation, which indicates an operator application implicitly by simple juxtaposition of the operator’s arguments, forgoing explicit rendering of the operator’s symbol as a prefix, infix, or suffix. For reason of denotational style, these operators will be called products despite that they suggest sums.

**Definition A.16.** Let $\Theta$ and $\Phi$ be disjoint (definition A.3) ensembles. The dyadic product $(\Theta, \Phi) \mapsto \Theta \Phi$ is:

$$\Theta \Phi = \Theta \cup \Phi.$$
An immediate consequence of this definition is commutativity $\Theta \Phi = \Phi \Theta$, since $\Theta \cup \Phi = \Phi \cup \Theta$.

**Theorem A.17.** Let $\Theta$ and $\Phi$ be disjoint ensembles. Their dyadic product $\Upsilon = \Theta \Phi$ is an ensemble with domain $(\text{dom } \Theta \cup \text{dom } \Phi)$ and range $(\text{ran } \Theta \cup \text{ran } \Phi)$.

**Proof.** Suppose $i \in \text{dom } \Theta \Phi$. By definition A.16, $\Theta \Phi = \Theta \cup \Phi$. From this it follows that $i \in \text{dom } \Theta \cup \text{dom } \Phi$. Therefore, $i \in \text{dom } \Theta$, $i \in \text{dom } \Phi$, or both. The stipulation that $\Theta$ and $\Phi$ be disjoint entails $\text{dom } \Theta \cap \text{dom } \Phi = \emptyset$ (definition A.3). That stipulation eliminates the possibility of both memberships, leaving two feasible cases: either A) $i \in \text{dom } \Theta$ and $i \notin \text{dom } \Phi$, or B) $i \notin \text{dom } \Theta$ and $i \in \text{dom } \Phi$.

In case A, there exists $(i, \Theta_i) \in \Theta$, so $(i, \Theta_i) \in \Upsilon = \Theta \cup \Phi$. Furthermore, since $i \notin \text{dom } \Phi$, $\Upsilon_i = \Theta_i$ is well-defined. Since by definition A.19, each member of $\text{ran } \Theta$ is a non-empty set, then equivocally $\Upsilon_i = \Theta_i$ is a non-empty set.

Argumentation for case B, supporting that $\Upsilon_i = \Phi_i$, is well-defined and that $\Upsilon_i$ is a non-empty set, is obtained by interchanging the roles of $\Theta$ and $\Phi$ in case A.

Over all possibilities, $\Upsilon = \Theta \cup \Phi$ is well-defined as a mapping (family) and each element of its range is a non-empty set. Therefore the dyadic product of two disjoint ensembles is itself an ensemble.

The sub-case analysis establishes $\text{dom } \Theta \Phi \subseteq (\text{dom } \Theta \cup \text{dom } \Phi)$. For the converse, first assume $i \in \text{dom } \Theta$. Then there is some $P$ such that $(i, P) \in \Theta$. From this it follows that $(i, P) \in \Theta \cup \Phi$, so $i \in \text{dom } \Theta \Phi$ and $\text{dom } \Theta \subseteq \text{dom } \Theta \Phi$. Similarly, $\text{dom } \Phi \subseteq \text{dom } \Theta \Phi$. From these two inclusions it follows that $(\text{dom } \Theta \cup \text{dom } \Phi) \subseteq \text{dom } \Theta \Phi$. With the previous result that $\text{dom } \Theta \Phi \subseteq (\text{dom } \Theta \cup \text{dom } \Phi)$, we conclude $\text{dom } \Theta \Phi = (\text{dom } \Theta \cup \text{dom } \Phi)$.

The sub-case analysis shows also that for $(i, P) \in \Theta \Phi$, either $P \in \text{ran } \Theta$ or $P \in \text{ran } \Phi$. This implies that $\text{ran } \Theta \Phi \subseteq \text{ran } \Theta \cup \text{ran } \Phi$.

To demonstrate the converse, assume $P \in \text{ran } \Theta$. Then there exists $(i, P) \in \Theta$, and by definition A.16, $(i, P) \in \Theta \Phi$. From this we conclude that $\text{ran } \Theta \subseteq \text{ran } \Theta \Phi$. With modest changes as above we conclude $\text{ran } \Theta \Phi \subseteq \text{ran } \Theta \cup \text{ran } \Phi$.

Finally, with both $\text{ran } \Theta \Phi \subseteq \text{ran } \Theta \cup \text{ran } \Phi$ and its converse, $\text{ran } \Theta \Phi = \text{ran } \Theta \cup \text{ran } \Phi$. $\Box$

**Definition A.18.** Let $\Theta$ and $\Phi$ be disjoint ensembles. The dyadic space product of $\prod \Theta$ and $\prod \Phi$ is

$$
\prod \Theta \prod \Phi = \prod \Theta \Phi
$$

(that is, with $\Upsilon = \Theta \Phi$, the set of all $\Upsilon$-choices).

**Definition A.19.** Let $\Theta$ and $\Phi$ be disjoint ensembles generating choice spaces $\prod \Theta$ and $\prod \Phi$. Suppose $\alpha \in \prod \Theta$ and $\beta \in \prod \Phi$ are choices. Their dyadic choice product $(\alpha, \beta) \mapsto \alpha \beta$ is

$$
\alpha \beta = \alpha \cup \beta
$$

**Theorem A.20.** Let $\Theta$ and $\Phi$ be disjoint ensembles. The dyadic choice product is a bijection

$$
\prod \Theta \times \prod \Phi \leftrightarrow \prod \Theta \Phi
$$

**Proof.** Suppose $(\theta, \phi) \in \prod \Theta \times \prod \Phi$, whence it follows that choices $\theta \in \prod \Theta$ and $\phi \in \prod \Phi$. Consider the dyadic products $\theta \phi = \theta \cup \phi$ and $\Theta \Phi = \Theta \cup \Phi$. Since $\text{dom } \theta = \text{dom } \Theta = T$ and $\text{dom } \phi = \text{dom } \Phi = P$, then $\text{dom } \theta \phi = T \cup P = \text{dom } \Theta \Phi$. Let $i \in \text{dom } \theta \phi = T \cup P$. Since $\Theta$ and $\Phi$ are disjoint, then exactly one of two cases holds: either $i \in T$ and $i \notin P$, or $i \notin T$ and $i \in P$.

In the first case, $i \in T$, $\theta \phi(i) = \theta(i)$ and $\Theta \Phi(i) = \Theta(i)$. Since $\theta$ is a $\Theta$-choice, then $\theta(i) \in \Theta(i)$. But since $\theta \phi(i) = \theta(i)$ and $\Theta \Phi(i) = \Theta(i)$, then $\theta \phi(i) \in \Theta \Phi(i)$ for any $i \in T$. 

In the second case, $i \in P$, $\theta \phi(i) = \phi(i)$ and $\Theta \Phi(i) = \Phi(i)$. Since $\phi$ is a $\Phi$-choice, then $\phi(i) \in \Phi(i)$. But since $\theta \phi(i) = \phi(i)$ and $\Theta \Phi(i) = \Phi(i)$, then $\theta \phi(i) \in \Theta \Phi(i)$ for any $i \in P$. 

In both cases, we conclude $\theta \phi \in \Theta \Phi$. 

But for all $(\theta, \phi) \in \prod \Theta \times \prod \Phi$, $\theta \phi \in \Theta \Phi$. 

Therefore, $\prod \Theta \times \prod \Phi \leftrightarrow \prod \Theta \Phi$. 

$\Box$
Theorem A.23. Let ensemble Definition A.21. domain. In the above, degenerate case A.3.3. Choice subspaces. Any mapping, including a choice mapping, may be restricted to subsets of its domain.

Definition A.21. Let Ψ be an ensemble, and let $R \subseteq \text{dom } \Psi$ be a subset of its index set. Suppose $\chi \in \prod \Psi$ is a choice. A subchoice $\chi \mid R$ is the ordinary mapping restriction of $\chi$ to its domain subset $R$.

Remark. In the above, degenerate case $R = \emptyset$ yields $\chi \mid R = \emptyset$.

Definition A.22. Let Ψ be an ensemble. For each $R \subseteq \text{dom } \Psi$, the subspace $(\prod \Psi \mid R)$ is the set of subchoices $\{ \chi \mid R : \chi \in \prod \Psi \}$.

Theorem A.23. Let ensemble Ψ generate choice space $\prod \Psi$, and let $R \subseteq \text{dom } \Psi$ be a subset of its index set. The restriction of the choice space equals the choice space of the restriction:

$$\left( \prod \Psi \mid R \right) = \prod (\Psi \mid R).$$

Proof. Suppose $\xi \in (\prod \Psi \mid R)$. By definition A.21 there exists $\chi \in \prod \Psi$ such that $\xi = \chi \mid R$. By definition of Cartesian product, for each $i \in \text{dom } \Psi$, $\chi(i) \in \Psi(i)$. Since $R \subseteq \text{dom } \Psi$, then for each $r \in R$, $\xi(r) \in \Psi(r)$. Consider $\Psi \mid R$, for which $\text{dom } (\Psi \mid R) = R$. By definition of restriction, for $r \in R$, $(\Psi \mid R)(r) = \Psi(r)$. Since $\xi(r) \in \Psi(r)$ and $\Psi(r) = (\Psi \mid R)(r)$, then for any $r \in R$, $\xi(r) \in (\Psi \mid R)(r)$ – that is, $\xi$ is a choice of $\Psi \mid R$. From the preceding, $\xi \in (\prod \Psi \mid R)$ implies $\xi \in \prod (\Psi \mid R)$, or $(\prod \Psi \mid R) \subseteq \prod (\Psi \mid R)$.

Next suppose $\xi \in \prod (\Psi \mid R)$. Then, by definitions A.9 and A.10 covering Cartesian products, for each $r \in R$, $\xi(r) \in (\Psi \mid R)(r)$. The ensemble Ψ coincides with its restriction $\Psi \mid R$ on $R$. A restatement of this is
\((\Psi | R)(r) = \Psi(r)\) for \(r \in R\). Substituting \(\Psi(r)\) for \((\Psi | R)(r)\) yields \(\xi(r) \in \Psi(r)\) for each \(r \in R\). From this it follows that \(\xi \in (\prod \Psi) \mid R\), with the further implication that \(\prod (\Psi | R) \subseteq (\prod \Psi) \mid R\).

We conclude equality \((\prod \Psi) \mid R = \prod (\Psi | R)\) after establishing that each of these two sets is a subset of the other. \(\square\)

**Lemma A.24.** Let \(\Psi\) and \(\Phi\) be ensembles. If \(\prod \Phi\) is a subspace of \(\prod \Psi\), then \(\Phi \subseteq \Psi\).

**Proof.** Let \(\prod \Phi\) be a subspace of \(\prod \Psi\). By definition [A.22] there exists \(R \subseteq \text{dom } \Psi\) such that \(\prod \Phi = (\prod \Psi) \mid R\). By Theorem [A.23] the restriction of the choice space equals the choice space of the restriction: \((\prod \Psi) \mid R = \prod (\Psi | R)\). Transitivity of equality implies \(\prod \Phi = \prod (\Psi | R)\). Then, by Theorem [A.13] (invertibility of the Cartesian product), \(\Phi = \Psi | R\).

Suppose term \((i, P) \in \Phi\). Since \(\Phi = \Psi | R\), then \((i, P) \in \Psi | R\). By the definition of restriction, this implies both \((i, P) \in \Psi\) and \(i \in R\). Since \((i, P) \in \Phi\) implies \((i, P) \in \Psi\), we conclude \(\Phi \subseteq \Psi\). \(\square\)

**Lemma A.25.** Let \(\Psi\) and \(\Phi\) be ensembles. If \(\Phi \subseteq \Psi\) and \(R = \text{dom } \Phi\), then \(\Phi = \Psi | R\).

**Proof.** Consider \((i, P) \in \Psi | R\). It then follows from the definition of restriction that \((i, P) \in \Psi\) and \(i \in R\). But \(R = \text{dom } \Phi\), so \(i \in \text{dom } \Phi\). This implies there exists \((i, Q) \in \Phi\). Since \(\Phi \subseteq \Psi\), then \((i, Q) \in \Psi\). Since \(\Psi\) is a mapping, then \((i, P) \in \Psi\) and \((i, Q) \in \Psi\) implies \(P = Q\). From \(P = Q\) and \((i, Q) \in \Phi\), we infer that \((i, P) \in \Phi\). Thus \((i, P) \in \Psi | R\) implies \((i, P) \in \Phi\), so \(\Psi | R \subseteq \Phi\).

Next suppose \((i, P) \in \Phi\). From this it follows that \(i \in R = \text{dom } \Phi\). From the premises \((i, P) \in \Phi\) and \(\Phi \subseteq \Psi\) we conclude \((i, P) \in \Psi\). Together \((i, P) \in \Psi\) and \(i \in R\) imply that \((i, P) \in \Psi | R\). Thus \((i, P) \in \Phi\) implies \((i, P) \in \Psi | R\), so \(\Phi \subseteq \Psi | R\).

From \(\Psi | R \subseteq \Phi\) and \(\Phi \subseteq \Psi | R\) we infer \(\Phi = \Psi | R\). \(\square\)

**Lemma A.26.** Let \(\Psi\) and \(\Phi\) be ensembles. If \(\Phi \subseteq \Psi\), then \(\prod \Phi\) is a subspace of \(\prod \Psi\).

**Proof.** Set \(R = \text{dom } \Phi\). Since \(\Phi \subseteq \Psi\) by hypothesis, then by applying lemma [A.25] we infer \(\Phi = \Psi | R\). With this equality and Theorem [A.13] (invertibility of the Cartesian product), we have \(\prod \Phi = \prod (\Psi | R)\). Transitivity of equality implies \(\prod \Phi = (\prod \Psi) \mid R\). This last equality is exactly the premise of definition [A.22] \(\prod \Phi\) is a subspace of \(\prod \Psi\). \(\square\)

**Theorem A.27.** Let \(\Psi\) and \(\Phi\) be ensembles. \(\prod \Phi\) is a subspace of \(\prod \Psi\) if and only if \(\Phi \subseteq \Psi\).

**Proof.** Lemma [A.24] asserts that if \(\prod \Phi\) is a subspace of \(\prod \Psi\), then \(\Phi \subseteq \Psi\). Lemma [A.26] asserts that if \(\Phi \subseteq \Psi\), then \(\prod \Phi\) is a subspace of \(\prod \Psi\). This pair of converse implications establishes the biconditional. \(\square\)

**Lemma A.28.** If \(\Upsilon, \Psi,\) and \(\Phi\) are ensembles such that \(\Upsilon = \Psi \cup \Phi\), then \(\Psi \subseteq \Upsilon\) and \(\Phi \subseteq \Upsilon\).

**Proof.** Since \(\Upsilon\) is the dyadic product of \(\Psi\) and \(\Phi\), then by definition [A.16] \(\Psi\) and \(\Phi\) are disjoint ensembles and \(\Upsilon = \Psi \cup \Phi\).

Suppose \(i \in \text{dom } \Upsilon = \text{dom } (\Psi \cup \Phi)\). Through definition [A.3] disjointness entails that \(\text{dom } \Psi \cap \text{dom } \Phi = \emptyset\). Thus, if \(i \in \text{dom } \Upsilon\), exactly one of two cases hold: either A: \(i \in \text{dom } \Psi\) and \(i \notin \text{dom } \Phi\), or B: \(i \in \text{dom } \Phi\) and \(i \notin \text{dom } \Psi\).

Assume case A, that \(i \in \text{dom } \Psi\) and \(i \notin \text{dom } \Phi\). With \(\Upsilon = \Psi \cup \Phi\), it follows from the definition of set union that for any \(i \in \text{dom } \Psi, (i, P) \in \Psi\) implies \((i, P) \in \Upsilon\) — that is, \(\Psi \subseteq \Upsilon\).

For case B, similar argument leads to \(\Phi \subseteq \Upsilon\). \(\square\)
Corollary A.29. If \( \Upsilon, \Psi, \) and \( \Phi \) are ensembles such that \( \Upsilon = \Psi \Phi \), then \( \Psi(i) = \Upsilon(i) \) for \( i \in \text{dom } \Psi \), and \( \Phi(j) = \Upsilon(j) \) for \( j \in \text{dom } \Phi \).

Proof. Under identical premises, lemma \([A.28]\) provides \( \Psi \subseteq \Upsilon \) and \( \Phi \subseteq \Upsilon \). Suppose \( i \in \text{dom } \Psi \). If \( (i, P) \in \Psi \), then \( (i, P) \in \Upsilon \) since \( \Psi \subseteq \Upsilon \). The notation \( \Psi(i) = \Upsilon(i) \) (both equaling \( P \)) is equivalent. A similar argument demonstrates \( \Phi(j) = \Upsilon(j) \) for \( j \in \text{dom } \Phi \).

Theorem A.30. Let \( \Upsilon, \Psi, \) and \( \Phi \) be ensembles such that \( \Upsilon = \Psi \Phi \). For each \( v \in \prod \Upsilon \), there exist unique \( \psi \in \prod \Psi \) and \( \phi \in \prod \Phi \) such that \( v = \psi \phi \).

Proof. Suppose \( v \in \prod \Upsilon \). Since any choice has the same domain as its generating ensemble, \( \text{dom } \Upsilon = \text{dom } v \). Theorem \([A.17]\) states that \( \text{dom } \Upsilon = \text{dom } \Psi \cup \text{dom } \Phi \), from which transitivity of equality provides \( \text{dom } v = \text{dom } \Psi \cup \text{dom } \Phi \).

From lemma \([A.28]\) we conclude \( \Psi \subseteq \Upsilon \) and \( \Phi \subseteq \Upsilon \). Since these relations hold for entire ensembles, then the same is true of the ensembles’ domains: \( \text{dom } \Psi \subseteq \text{dom } \Upsilon \) and \( \text{dom } \Phi \subseteq \text{dom } \Upsilon \). By substitution, the previous result \( \text{dom } \Upsilon = \text{dom } v \) then establishes that \( \text{dom } \Psi \subseteq \text{dom } v \) and \( \text{dom } \Phi \subseteq \text{dom } v \).

The inclusion \( \text{dom } \Psi \subseteq \text{dom } v \) ensures that the restriction \( \psi = v \mid \text{dom } \Psi \) is well-defined. Similarly \( \phi = v \mid \text{dom } \Phi \) is also well-defined.

We next focus on the restriction \( \psi \) constructed above, seeking to demonstrate that it is also a member of the choice space \( \prod \Psi \). Suppose term \( (i, p) \in \psi \). Since \( \psi = v \mid \text{dom } \Psi \), then both \( (i, p) \in v \) and \( i \in \text{dom } \Psi \). Since \( v \in \prod \Upsilon \) by hypothesis, definition \([A.9]\) demands that \( p \in \Upsilon(i) \) whenever \( (i, p) \in v \). Corollary \([A.29]\) asserts \( \Psi(i) = \Upsilon(i) \) for \( i \in \text{dom } \Psi \). Since \( p \in \Upsilon(i) \) and \( \Upsilon(i) = \Psi(i) \) then \( p \in \Psi(i) \). Thus for any \( (i, p) \in \psi \), it follows that \( p \in \Psi(i) \). This means that \( \psi \) is a choice of \( \Psi \) by definition \([A.9]\)—that is, \( \psi \in \prod \Psi \) by definition \([A.10]\).

Similar reasoning establishes that the other restriction \( \phi \) is a member of \( \prod \Phi \). The unique \( \psi \in \prod \Psi \) and \( \phi \in \prod \Phi \) such that \( v = \psi \phi \) are expressed by the restrictions \( \psi = v \mid \text{dom } \Psi \) and \( \phi = v \mid \text{dom } \Phi \).

A.4. Persistent-volatile partition. Definition \([3.1]\) asserts that basis \( \langle \Psi, \Phi \rangle \) is comprised of two ensembles satisfying \( \Phi \subseteq \Psi \). This allows partitioning terms of \( \Psi \) into two sets: those terms that are members of both \( \Psi \) and \( \Phi \), and those terms that are members of \( \Psi \) but not \( \Phi \). The ensemble difference terminology of definition \([A.5]\) poses the minuend, subtrahend, and remainder of this basis as respectively \( \Psi \), \( \Phi \), and \( \Psi \setminus \Phi \).

Terminology. This partition is important when interpreted as systems theory. The minuend \( \Psi \) generates a choice space \( \prod \Psi \) called the stimulus space. The subtrahend \( \Phi \) generates the persistent (alternatively response) space \( \prod \Phi \). The remainder \( \Psi \setminus \Phi \) generates the event space \( \prod (\Psi \setminus \Phi) \).

Theorem A.31. Let \( \langle \Psi, \Phi \rangle \) be a basis. The persistent \( \Phi \) and volatile \( \Upsilon = \Psi \setminus \Phi \) generating ensembles are disjoint and complementary with respect to the generating ensemble \( \Psi \) of the stimulus space. Expressed in dyadic product,

\[ \Psi = \Phi \Upsilon \]

Proof. Since \( \langle \Psi, \Phi \rangle \) is a basis, then \( \Phi \subseteq \Psi \) by definition \([3.1]\). With that result and by lemma \([A.6]\) the subtrahend \( \Phi \) and remainder \( \Psi \setminus \Phi \) are disjoint and complementary with respect to \( \Psi \).

The dyadic product recapitulates these results. Since \( \Psi \setminus \Phi \) and \( \Phi \) are disjoint the dyadic product \( \left[ \Psi \setminus \Phi \right][\Phi] \) is well-defined. By definition \([A.16]\) \( \left[ \Psi \setminus \Phi \right][\Phi] = (\Psi \setminus \Phi) \cup \Phi = \Psi \).

Remark. The case \( \Phi = \emptyset \) does not occur naturally in systems theory because no proper system is unresponsive to all possible stimuli. When \( \Psi = \Phi \), the basis has no event space through which to receive transient external stimuli.
APPENDIX B. UNCOVERABLE PROCESSES

Although any procedure does cover some process, some processes have no covering procedure. This disparity arises naturally through limiting the quantity of distinct functionalities participating in a procedure. Here the constraining mechanism is the catalog of functionality, whose membership must be finite. This stricture’s rationale is to emulate software, which is presumed to possess finite functionality.

Uncoverability of a process entails more than failure of definition 4.6 in the case of a particular procedure; uncoverability implies failure for any procedure constructed from a given catalog of functionality. With process \( \{ f_n \} : \mathbb{N} \rightarrow \prod \Psi \times \prod \Phi \) and catalog of functionality \( \mathcal{F} \), uncoverability requires an \( i \in \mathbb{N} \) and term \( f_i = f \) such that \( f \notin f \) for each \( f \in \mathcal{F} \).

The pigeonhole principle can verify uncoverability, but not coverability. Suppose two frames have the same abscissa but different ordinates. No single functionality can cover both frames, since functionalities are mappings. In more general analogy, let distinct equi-abscissa frames be pigeons, while functionalities be pigeonholes. If more than \( N \) pigeons occupy \( N \) pigeonholes, then some pigeonhole contains more than one pigeon, which is not allowed.

The frame set derived from the initial segment of length \( k \) of process \( \{ f_n \} \) is the set \( F = \{ f : f = f_i \text{ and } i \leq k \} \). This set’s \( \psi \)-homogeneous subset contains only those frames having initial condition (abscissa) \( \psi \in \prod \Psi \).

The corresponding end-condition set consists of those frames’ ordinates. Obviously the \( \psi \)-homogeneous end-condition set must have cardinality \( \left| \{ \phi : (\psi, \phi) = f_i \text{ and } i \leq k \} \right| \leq k \). The limit supremum (respecting initial segment length and homogeneity choice) presents the process’ worst case scenario.

**Definition B.1.** Let \( (\Psi, \Phi) \) be a basis for process \( \{ f_n \} : \mathbb{N} \rightarrow \prod \Psi \times \prod \Phi \) and catalog of functionality \( \mathcal{F} \). Catalog \( \mathcal{F} \) under-pigeonholes process \( \{ f_n \} \) if

\[
|\mathcal{F}| < \lim_{n \to \infty} \left( \sup_{\psi \in \prod \Psi} \left( \left| \{ \phi : (\psi, \phi) = f_i \text{ and } i \leq n \} \right| \right) \right).
\]

Whenever the limit supremum fails to converge, no procedure based on a (finite) catalog can cover the process.

APPENDIX C. MIL-STD-882 AND THE CPP

MIL-STD-882 is the United States Department of Defense Standard Practice for System Safety. Revision E became effective May 11, 2012. In preference to accident, this standard prefers the term mishap, which it defines as “an event or series of events resulting in unintentional death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.”

Its safety risk assessment method uses the compound Poisson process (CPP) to represent the timing and severity of mishaps. MIL-STD-882E partitions compound Poisson processes into a lattice of categories and levels that covers the range of interest. The category is a variable which, in an explicit range [1-4], expresses the expectation \( (\mu_L) \) of the CPP loss random variable \( L \). The level is a variable which, in an explicit range [A-F], expresses the rate or intensity \( \lambda \) of the CPP.

The system of categories and levels agrees with the limits of discernibility of human intuition. Two different compound Poisson processes having the same category and level are indeed different but in practice are indistinguishable. This characteristic imposes a logarithmic organization on the categories and levels.
Table 2 above is a qualitative description of levels. Table 3 below, appearing in MIL-STD-882E Appendix A, outlines certain pitfalls in accomplishing the same task quantitatively. Numerical expression of the intensity or rate of occurrence is generally preferable to mere qualitative phrasing. For quantitative description, the intensity is the ratio of mishaps (numerator) to some measure of exposure (denominator).
| Description | Level | Individual Item | Fleet/Inventory* | Quantitative |
|-------------|-------|-----------------|-----------------|--------------|
| Frequent    | A     | Likely to occur often in the life of an item | Continuously experienced. | Probability of occurrence greater than or equal to $10^{-1}$. |
| Probable    | B     | Will occur several times in the life of an item | Will occur frequently. | Probability of occurrence less than $10^{-1}$ but greater than or equal to $10^{-2}$. |
| Occasional  | C     | Likely to occur sometime in the life of an item | Will occur several times. | Probability of occurrence less than $10^{-2}$ but greater than or equal to $10^{-3}$. |
| Remote      | D     | Unlikely, but possible to occur in the life of an item | Unlikely but can reasonably be expected to occur. | Probability of occurrence less than $10^{-3}$ but greater than or equal to $10^{-6}$. |
| Improbable  | E     | So unlikely, it can be assumed occurrence may not be experienced in the life of an item | Unlikely to occur, but possible. | Probability of occurrence less than $10^{-6}$. |
| Eliminated  | F     | Incapable of occurrence within the life of an item. This category is used when potential hazards are identified and later eliminated. | | |

Table 5. MIL-STD-882E Example Probability Levels

* The size of the fleet or inventory should be defined.

The false hegemony of a single intuitively understood measure of exposure will now be examined. We will find that, however well-intended, Table 3 lacks essential explanation. Without that, it is an oversimplification.

“Natural” measures of exposure must embrace a variety of units, some examples of which are: the life of one item, number of missile firings, flight hours, miles driven, or years of service. For example, an exposure measure of miles driven is expected silently to exclude substantial periods when the system is out of use. Similar would be any situation-based measure of exposure having a sizable portion of time spent in unused status (time not counted). This topic appeared in the intermittent compound Poisson process. The natural unit of exposure can be tuned to the culture of a particular hazard. However, lacking conversion capability, this freedom of choice leads to the problem of a system composed of a heterogeneous plethora of non-comparable exposure units.

What is behind this incomparability? Natural units are important but incomplete – MIL-STD-882E needs additional factors to paint a full quantitative picture. There is need for conversion of various natural units into a single common standard unit, so that comparison involves only observation of magnitudes, without pondering the meaning of different units. This is particularly important in the cases of many ambiguous references to “life.” Suppose we arbitrarily standardize time duration at one year. We then define a conversion factor $p$, which means that $p$ years constitute a life. A measure $\iota$ quantifies what fraction of time the system’s mission is inactive or idle. A conversion factor for remaining units must be established; without specifying what units remain to be converted, we can say that the unit conversion calculus of elementary physics results in some linear coefficient $\kappa$. With $N$ a natural exposure unit and $U$ a standard measure, what we have stated so far is summarized in the following form:

$$U = \frac{\kappa \cdot (1 - \iota)}{p} \cdot N.$$  

Standard units measure statistical risk as resulting from exposure to an intermittent compound Poisson process. These standard units may not be a proper exposure, but measure the exposure expected in a year’s duration. For this reason we celebrate the importance of the role of pure natural units; it is important to
understand risk as proportionate to exposure. To understand this importance, imagine yourself as the one exposed to a transient but intense hazard. But that does not imply the dismissal of statistical risk as a concern; it is also part of the risk analysis picture to consider how much risk exposure occurs within a given duration. This is the role of the standard unit.

Another complicating factor is the use of the term “level” itself. A level is a designator for a class of possibly intermittent indistinguishable probability distributions. Rather than being clear about this, MIL-STD-882E equivocates greatly in Table 3, confusing this designator with a literal probability statement. Only after full quantitative analysis is completed ($p$, $i$, and $\kappa$ known) can definite statements concerning probability be asserted. It is insufficient to mandate vague documentation of “all numerical definitions of probability used in risk assessments” without further guidance.

Table 4 below is a categorical rendering of the hyperbola of statistical risk. Definition [11.3] asserts $h = \lambda\mu_L$. Excepting the administrative level “Eliminated”, this cross-tabulation presents the level ($\lambda$) on the vertical axis and the category ($\mu_L$) along the horizontal axis. For each combination of level and category, another categorical variable $12$ represents the statistical risk $h = \lambda\mu_L$ with values: High, Serious, Medium, and Low.

| SEVERITY / PROBABILITY | Catastrophic (1) | Critical (2) | Marginal (3) | Negligible (4) |
|------------------------|-----------------|--------------|--------------|----------------|
| Frequent (A)           | High            | High         | Serious      | Medium         |
| Probable (B)           | High            | High         | Serious      | Medium         |
| Occasional (C)         | High            | Serious      | Medium       | Low            |
| Remote (D)             | Serious         | Medium       | Medium       | Low            |
| Improbable (E)         | Medium          | Medium       | Medium       | Low            |
| Eliminated (F)         | Eliminated      |              |              |                |

**TABLE 6. MIL-STD-882E Risk Assessment Matrix**

This table suffers the same ambiguity as in Table 3. MIL-STD-882’s definitions are clearly inadequate for quantitative analysis. Through equivocation, exposure to an intermittent compound Poisson process is regarded as not different than exposure to a compound Poisson process, despite that the difference becomes obvious through the linear factor $(1 - i)$. MIL-STD-882 is an evolving document in its fifth major revision; let us hope these ambiguities are resolved in the future.

**APPENDIX D. OTHER APPROACHES TO AUTOMATA**

D.1. Deterministic finite automaton. This depiction of the deterministic finite automaton [8] appears in Wikipedia:

An automaton is represented formally by the 5-tuple $(Q, \Sigma, \delta, q_0, A)$, where:

- $Q$ is a finite set of states.
- $\Sigma$ is a finite set of symbols, called the alphabet of the automaton.
- $\delta$ is the transition function, that is, $\delta : Q \times \Sigma \rightarrow Q$.
- $q_0$ is the start state, that is, the state which the automaton is in when no input has been processed yet, where $q_0 \in Q$.
- $A$ is a set of states of $Q$ (i.e. $A \subseteq Q$) called accept states.

An approach for engineers is found in [4].

12Not to be confused with the categorical variable named “category”
REFERENCES

[1] Benjamin S. Blanchard and Wolter J. Fabrycky, Systems Engineering and Analysis, 3rd edition, Prentice Hall, 1998
[2] Arthur D. Hall, A Methodology For Systems Engineering, (New York: Van Nostrand Reinhold Company), 1962
[3] John D. Musa and Anthony Iannino and Kazuhiro Okumoto, Software reliability - measurement, prediction, application, McGraw-Hill, 1987
[4] Hopcroft, J. E. and Ullman, J. D., Introduction to Automata Theory, Languages and Computation, Second Edition, Addison-Wesley, 1979
[5] Paul R. Halmos, Naive Set Theory, Springer-Verlag New York, 1974
[6] Department of Defense, Joint Software Systems Safety Engineering Handbook, Internet, http://www.acq.osd.mil/se/docs/Joint-SW-Systems-Safety-Engineering-Handbook.pdf, retrieved 2014-09-19
[7] Department of Defense, MIL-STD-882E (Standard Practice for System Safety), Internet, http://www.assistdocs.com/search/search__basic.cfm, retrieved 2012-05-13T18:01Z
[8] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Automata_theory, retrieved July 18, 2011
[9] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Emergence, retrieved June 13, 2014
[10] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Frequentist_probability, retrieved March 10, 2014
[11] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Poisson_distribution, retrieved May 20, 2013
[12] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Poisson_process, retrieved May 20, 2013
[13] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Law_of_total_expectation, retrieved May 20, 2013
[14] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Compound_Poisson_process, retrieved May 20, 2013
[15] Wikipedia®, Internet, http://en.wikipedia.org/wiki/Cumulant, retrieved May 24, 2013