

Recovering Process Boundaries from Complete Event Sequences in Finite Workflow Models

Wanhong HUANG
huangwanhong@serendip.ngo

Abstract

A complete ordered event record can leave the boundaries between named processes unresolved. This paper examines that problem in finite workflow models where each process type has a nonempty event pattern. It distinguishes grouped histories, flattened event words, composite effects, and evidence of actual occurrence. Exact process-type recovery is characterized by injectivity of the flattening map on a declared admissible language. For unrestricted concatenations, the paper reconstructs the classical Sardinas–Patterson residual criterion with explicit witness invariants, soundness, completeness, and termination arguments. A fictional document-handling dossier shows that two different process decompositions can preserve the same total process count, making recovery query-specific. A further example separates unique decoding of completed finite records from boundary identification during an ongoing stream. The discussion examines delimiters, identifiers, constrained histories, uncertain coverage, and competing criteria of process unity. Verified antecedents in coding, process mining, provenance representation, and experiential philosophy locate the proposal without a claim of conceptual priority. The formal results remain conditional on the vocabulary and record convention; historical truth, institutional boundary legitimacy, and responsibility require additional grounds. Open questions concern constrained languages, closure-sensitive inference, and comparison of process-individuation criteria.

Keywords: process boundaries; event sequences; finite workflows; unique decoding; historical reconstruction; process individuation

Research Purpose and Scope

This note states the purpose of circulation, the status of the argument, and the role of research assistance. The paper presents current insights for criticism and wider discussion in public and scholarly settings. It makes no claim to conceptual priority. Similar concepts and arguments may already exist in other disciplines, languages, and intellectual traditions; further comparison may alter the terminology, scope, or contribution.

Definitions organize a bounded inquiry. Conditional formal results carry their stated assumptions, and illustrative cases support conceptual analysis. Empirical validation, wider applicability, and the adequacy of proposed normative premises remain open where the main text identifies those obligations. Relational analysis supplies a method of investigation while stronger ontological questions remain suspended.

AI assistance contributed to source discovery and pre-use verification, argumentative development, formal checking, organization, and drafting. The accompanying project records distinguish retained research insights, newly proposed analysis, and verified literature. The manuscript is prepared for the author's review; circulation and submission decisions remain with the author. Critical responses concerning antecedents, assumptions, counterexamples, and alternative formulations are welcome.

DRAFT

1 Introduction

This paper examines the recovery of process boundaries from complete event sequences in finite workflow models. Its objective is to identify which grouping information can remain absent even when event order is preserved, and which historical questions can nevertheless be answered. The argument connects a process-block representation to classical unique-decipherability methods, develops a fictional workflow dossier, and examines the additional evidence needed for record closure, process individuation, and historical warrant.

A composite mechanism can preserve an endpoint while losing the sequence of operations that produced it. Retaining the full event sequence repairs that particular loss only when the historical question concerns the ordered events themselves. A further question remains: which events belong together as one process? A record can preserve every event position and its order while leaving the boundaries between named routines unresolved. The missing information then concerns grouping, rather than the existence of an earlier or later event within the record.

Consider three observed event positions with types a, b, a . A workflow vocabulary may contain a routine producing a , a routine producing ab , and a routine producing ba . The record can then represent a first routine followed by a third, or a second routine followed by a first. Both presentations preserve the same event order. They disagree about the internal process boundary and about which routine contains the event of type b . A claim that the record is complete therefore needs a target: completeness of the event sequence can coexist with incomplete evidence of its process decomposition.

The paper studies that problem through a finite codebook. Each named process type has a nonempty finite event word, and a process history concatenates such blocks. This representation permits an exact recovery question. A complete event word determines the process-type history when the flattening map is injective on the declared admissible histories. For unrestricted concatenations, the problem has classical antecedents in unique decipherability. The paper reconstructs a finite residual test and its proof, with the aim of making the assumptions accessible to a philosophical and interdisciplinary discussion.

The formal result is useful partly because it separates several conclusions. A record may have a unique process decomposition, several decompositions, or none within the declared model. Several decompositions can agree on a particular query, such as the total number of process blocks, while disagreeing on their types or boundaries. A unique decomposition of a completed finite record can also coexist with an inability to identify its first block before the record closes. These distinctions prevent an undifferentiated claim that a history is either fully recoverable or entirely lost.

The analysis retains a further separation between mathematical recoverability and historical truth. A decoder operates on a record and an admissible-history model. Its output establishes what those inputs jointly determine. Evidence is still needed that the recorded events occurred, that relevant events were retained, that the vocabulary fits the activity, and that the record includes complete blocks rather than a cut through an ongoing process. A unique formal answer can inherit an error from any of those premises.

The process boundary itself can also be contested. An institution may delimit a routine by a procedural rule, a participant may understand a continuing undertaking across several administrative episodes, and an analyst may group events for a particular explanatory purpose. Such criteria can overlap while answering different questions. The present codebook supplies one bounded procedural criterion. It leaves a broader account of process unity open, including experiential integration, causal continuity, and continuing responsibilities.

The contribution therefore has both a formal and an interpretive scope. Formally, it supplies explicit conditions, a stepwise residual argument, a query-specific recovery example, and a closure-sensitive counterexample. Interpretively, it explains how process grouping can be lost before the familiar compression from event word to composite effect, and why additional identifiers or boundary markers remain assertions requiring provenance. The underlying coding results are established antecedents, and the paper makes no claim to their priority.

The exposition begins with verified sources in coding theory, process mining, provenance representation, and a limited philosophical comparison concerning experiential unity. It then defines the process

representation and the recovery target before introducing the residual procedure. The workflow dossier and record-closure analysis make the formal distinctions concrete. The final sections consider record design, competing process criteria, objections, and open research obligations. Throughout, stronger ontological conclusions remain suspended and the illustrative cases remain explicitly fictional.

2 Antecedents in Coding, Provenance, and Process Unity

This section locates the paper’s problem in verified prior work. Its objective is to distinguish inherited mathematical methods from the present process-history application and to identify close approaches that already question event grouping. The discussion considers unique decipherability, case-based and object-centric process mining, provenance representation, and Dewey’s account of experiential integration.

Sedgewick and Wayne’s data-compression booksite distinguishes uniquely decipherable codes from prefix-free codes and presents the Sardinas–Patterson test through residual suffixes. It includes finite examples in which a code is uniquely decipherable despite failing prefix freedom [2]. The present paper uses these established ideas to analyze process-block recovery. Its residual construction states the allowed comparisons explicitly and supplies a proof. The code words represent event patterns in a stipulated workflow, so their practical meaning requires assumptions beyond the mathematical decoding property.

Van der Aalst, Adriansyah, and van Dongen’s selected case-based treatment begins from event records associated with an activity and a process instance. It studies alignment between an event log and a process model for conformance and performance analysis [3]. This antecedent makes the role of case assignment visible. The present question concerns a situation in which the boundaries or process assignments themselves are among the unknowns. That situation is a different starting point from one in which the relevant case relation has already been supplied.

Closer attention to grouping appears in van der Aalst’s object-centric process-mining account. It considers several possible case notions and events related to multiple objects, and examines distortions introduced by flattening such relations into a single case view [4]. The present sequential codebook is much narrower. It studies contiguous, nonoverlapping blocks in one ordered record. The comparison prevents the codebook problem from being presented as the first recognition that event-to-process relations require scrutiny; it also identifies overlapping object relations among the main extensions that the current model leaves open.

The W3C PROV data model provides another relevant representation. It assigns identifiers to activities, permits start and end times and attributes, and supplies relations concerning the start and end of an activity [5]. Such structures can retain boundary assertions that a flat event word omits. Their presence makes the asserted grouping explicit. Whether the assertion accurately identifies an actual process remains a question about evidence and interpretation, which the present paper treats separately from the availability of a data vocabulary.

Dewey’s chapter “Having an Experience” offers a limited philosophical comparison. It distinguishes integrated, individually qualified experience from dispersed succession and interruption, while preserving differentiated phases within a whole. The chapter also allows that an experientially integrated course can have a harmful outcome [1]. This comparison directs attention to forms of unity that a sequence of labels alone leaves open. The present administrative blocks have a different criterion: a failed, interrupted, or mechanically executed routine can still count as a process under its declared procedural rule.

The sources address different levels of the inquiry. Coding methods ask whether a string has one decomposition under a fixed vocabulary. Process-mining methods examine relations among observed events, cases, objects, and models. Provenance specifications provide ways to represent assertions about activities and their production. The philosophical comparison concerns an experienced unity that can exceed a formal boundary convention. Keeping these roles distinct permits the paper to draw on their insights without equating their objects.

The epistemological foundation is correspondingly layered. A mathematical recovery claim depends on a representation and a target. A historical claim depends additionally on evidence that the representation fits what occurred. A process-individuation claim depends on a criterion for grouping the relevant

activity. A normative claim about responsibility or institutional treatment requires reasons that connect the grouped activity to the proposed judgment. A result at one level can inform another while leaving its additional premises open.

The paper's own proposal concerns the movement among those levels in a bounded setting. It makes grouping loss explicit before composition, identifies exact recovery conditions under a finite process vocabulary, and uses the resulting distinctions to examine record closure and institutional boundary conventions. Wider comparison may reveal closer mathematical or philosophical precedents. The current aim is to make the inquiry and its assumptions available for criticism and further development.

3 Process Presentations and Successive Information Loss

This section defines the levels of process representation used in the argument. Its objective is to separate a grouped history, an event sequence, a composite effect, and a claim about actual occurrence. The discussion introduces the event and process alphabets, relates flattening to the repository's earlier composition distinction, and states the assumptions needed for the finite recovery model.

Let Σ be a finite event-type alphabet and let K be a finite nonempty set of named process types. A word in Σ^* gives an ordered sequence of event types. Its positions distinguish repeated entries of the same type. Additional occurrence identifiers or timestamps can be attached to those positions, but the formal block code initially uses their types and order. An actual record requires evidence that its entries refer to the intended occurrences.

Assign to each process type $k \in K$ a nonempty finite event word $c(k) \in \Sigma^+$. A process-type history is a finite word $h = k_1 \cdots k_s$ in K^* . The history's positions distinguish repeated occurrences of the same process type within the model. Equation 3.1 defines the flattening map that removes the process boundaries while retaining the event word.

$$c^*(k_1 \cdots k_s) = c(k_1) \cdots c(k_s), \quad c^*(\epsilon) = \epsilon. \quad (3.1)$$

Equation 3.1 specifies concatenation in execution order. A process history supplies both the selected blocks and their order. The flattened word retains their combined event sequence while omitting the places where one block ends and another begins. The empty history represents zero blocks and produces an empty record under this convention.

Definition 3.1 (Finite block presentation). A finite block presentation consists of a process vocabulary K , an event alphabet Σ , a nonempty-word assignment c , and an admissible-history language $L \subseteq K^*$. A history in L is a proposed process-type decomposition. Its interpretation as an actual history requires separate provenance and domain adequacy.

The nonempty-word assumption performs substantive work. If a process type could generate no recorded event, arbitrarily many such blocks could be inserted without changing the event record. The finite codebook analysis would then fail to bound the number of possible blocks by record length. An application with silent activities must represent them through a different model or add appropriate evidence. Completeness with respect to the chosen event alphabet is weaker than an assurance that every consequential activity leaves a recorded event.

The language L states which process-type sequences the model admits. The unrestricted case is $L = K^*$. A workflow specification can impose a narrower language through order rules, initial conditions, or compatibility constraints. Whether an actual history satisfies those restrictions remains an evidential question. In particular, a rule about what participants should do supplies a normative specification; it requires additional support before being used as an exhaustive account of what they did.

The flattening map precedes further familiar compressions. If each event type denotes a mechanism in a declared action model, its ordered word can be mapped to a composite mechanism, that mechanism to its induced transformation, and the transformation to its endpoint at a selected initial configuration. Each map can merge distinctions retained at the preceding level. For an idempotent mechanism, one

application and two applications can have the same composite. A nonfaithful action can give different composite elements the same transformation, and different transformations can agree at one selected initial state.

Those later losses should remain distinct from boundary loss. Two grouped histories can flatten to the same event word before any multiplication occurs. Every subsequent function of that word then assigns them the same value. Retaining a composite, a transformation, or an endpoint alone cannot recover a grouping that was already absent from the complete flattened word, unless additional information enters through another record or restriction.

Claim 3.2 (Persistence of a flattening ambiguity). If distinct admissible process histories h and h' satisfy $c^*(h) = c^*(h')$, every record obtained solely by applying a function to that event word gives them the same recorded value.

Proof. Apply the same record function to the equal flattened words. Equality of the resulting records follows immediately. A decoder of that coarser record cannot assign different exact histories to the same input without using an additional premise or evidence source. $\square$

The claim leaves open recovery of a coarser historical target. Two histories may differ in boundaries while agreeing on their total number of blocks. They may disagree about the first process type while agreeing that a review occurred. The question of recoverability should therefore name its target before evaluating the record. A demand to recover the entire history is stronger than a demand to answer one well-specified query about it.

The presentation is also narrower than a general theory of processes. Blocks are sequential and nonoverlapping, each type has one fixed event pattern, and the record covers complete blocks under the declared convention. Actual processes can branch, overlap, repeat with variation, or remain unfinished. The model supplies an explicit baseline for assessing grouping loss. The later objections identify the additional structures required to represent those broader possibilities.

4 Boundary Recovery and Admissible Decoding Sets

This section states the exact recovery target before introducing an algorithm. Its objective is to distinguish recovery of a full process-type history from recovery of a boundary pattern or a particular historical query. The discussion defines admissible decoding sets, proves the injectivity criterion, and explains the special assumptions under which the classical unrestricted code problem applies.

For a recorded event word w , Equation 4.1 collects its admissible process-type histories.

$$\mathcal{D}_L(w) = \{h \in L : c^*(h) = w\}. \quad (4.1)$$

Equation 4.1 is a compatibility set. Its members are histories that satisfy the declared language and reproduce the event word exactly. The set assigns no probabilities, preferences, or historical endorsement to its members. Any ranking requires additional evidence or an explicitly introduced model of uncertainty.

Definition 4.1 (Exact process-type recovery). A block presentation permits exact process-type recovery on L when every event word in $c^*(L)$ has exactly one admissible decoding. Equivalently, the restriction of c^* to L is injective.

The equivalence follows directly from the definition of injectivity. If two admissible histories flatten to the same word, that word has at least two decodings. Conversely, two different decodings of a word exhibit a failure of injectivity. The formulation also gives an inverse on the image $c^*(L)$ when exact recovery holds. It leaves words outside that image without a model-consistent decoding.

An empty decoding set should therefore be interpreted carefully. It can indicate that the record contains an event pattern outside the model, that an event is missing or miscoded, that the observation window cuts through a block, or that the admissible-language restriction is too strong. It is a conflict

between the record and the declared presentation. Choosing one explanation requires an investigation of the relevant evidence.

A singleton decoding set has a different limit. It identifies the one history admitted by the current model for that word. If the language excludes an actual deviation or the record is inaccurate, the unique result can still misdescribe what happened. Mathematical uniqueness is a property of the record-model pair. Historical warrant includes the adequacy of both members of that pair.

Boundary recovery can be weaker than process-type recovery. A boundary pattern records the event positions at which blocks end. If two process types have the same codeword, exchanging their labels leaves the boundary pattern unchanged. Thus a record can determine the segmentation while failing to determine the named types. When the assignment c is injective, each complete segment matching a codeword determines its type; in that special case, unique segmentation and unique process-type recovery coincide.

Claim 4.2 (Immediate type collision). For the unrestricted language K^* , two distinct process types with the same assigned event word prevent exact process-type recovery, even when their one-block records have an unambiguous boundary pattern.

Proof. The two one-letter histories in K^* are distinct and flatten to the same event word. Their process labels differ, while each uses the entire record as one block. This witnesses type ambiguity without requiring a second boundary location. $\square$

The collision check should precede a test applied only to the set of distinct codewords. Passing from c to its image $C = c(K)$ removes duplicate words. That operation can conceal different process labels assigned to the same word. The classical code-set problem is therefore used below under the explicit assumption that c is injective. It then asks whether every finite concatenation of words from C has one decomposition into those words.

Recovery of a particular query is more flexible. Let $f : L \rightarrow A$ assign an answer to each admissible history. The record determines the answer exactly when f is constant on $\mathcal{D}_L(w)$. If the set contains several histories but all give the same answer, that query remains recoverable. If the answers differ, the image of the decoding set under f records the unresolved alternatives. The workflow dossier uses this distinction to separate total process count from boundary and type recovery.

Restricting the language can narrow the decoding set. If $L' \subseteq L$, then $\mathcal{D}_{L'}(w) \subseteq \mathcal{D}_L(w)$. This set inclusion is a mathematical consequence of adding constraints. Its historical interpretation depends on whether the added constraints are warranted. Excluding one plausible history because it violates the very rule under investigation would turn a conformance question into an assumption of conformance.

The residual method in the next section addresses the unrestricted finite-code case. A positive result implies uniqueness on every restricted language using that codebook, provided the code assignment remains injective. A negative result for unrestricted concatenations can leave a particular restricted language uniquely recoverable. The two questions should therefore be reported separately, with the unrestricted ambiguity witness available for checking whether the proposed restrictions actually exclude it.

5 A Finite Residual Construction for Block Codes

This section defines the residual procedure used to test unrestricted unique decipherability. Its objective is to make each comparison and stopping condition explicit before proving the result. The construction assumes a finite set of nonempty distinct codewords, retains unresolved suffixes between competing decompositions, and limits every new comparison to a reached residual and an original codeword.

Let $C \subseteq \Sigma^+$ be the image of an injective finite block assignment. A word u is a prefix of v when $v = ur$ for some word r ; it is a strict prefix when the remaining suffix r is nonempty. If two candidate first blocks can begin the same event record, one must be a prefix of the other. After cancelling their common initial events, the longer candidate leaves a residual suffix.

Equation 5.1 collects those initial nonempty residuals.

$$R_0 = \{r \in \Sigma^+ : \exists u, v \in C, u \neq v \wedge ur = v\}. \quad (5.1)$$

Equation 5.1 begins with distinct original codewords. If neither is a prefix of the other, their event patterns already disagree and they cannot begin two decompositions of the same record. If one is a strict prefix, the residual represents how far one encoded candidate currently extends beyond the other.

Suppose a reached residual is s . The shorter candidate history can append one original codeword $u \in C$. If u is a prefix of s , cancelling it leaves a shorter residual on the same side. If s is a prefix of u , cancelling s leaves a residual on the opposite side. If the words agree exactly, the residual becomes empty and the two candidate histories have reached the same encoded endpoint.

Equation 5.2 states the cumulative update while the empty residual has not yet been reached.

$$R_{j+1} = R_j \cup \{r \in \Sigma^* : \exists s \in R_j \exists u \in C, ur = s \vee sr = u\}. \quad (5.2)$$

Equation 5.2 retains earlier residuals and adds the suffixes produced by the two cancellation cases. A new step compares an original codeword with a residual. Arbitrary comparisons between two residuals are outside this rule, because a residual records an unfinished difference rather than an independently available process block.

The procedure stops with an ambiguity when ϵ appears. It stops with a unique-decipherability result when a stage adds nothing and ϵ remains absent. A reached residual that is itself a codeword provides an immediate route to the empty residual at the next comparison. An implementation can therefore flag that condition early while retaining the same mathematical meaning.

Definition 5.1 (Residual witness). A residual witness consists of two process-type histories with different first types, together with an orientation indicating which encoded history extends beyond the other by the residual word. Reaching the empty residual gives two distinct histories with the same event encoding.

The witness contains more information than the set used for the decision procedure. The set needs only to record which residuals are reachable, because either orientation can be interpreted symmetrically. For an inspectable ambiguity report, the implementation can also retain the predecessor residual, the codeword appended, and the orientation change. Following those records reconstructs the competing process histories.

The restriction to nonempty codewords makes the procedure finite. Every initial residual is a nonempty proper suffix of an original word. Before the empty residual appears, cancelling an original word from a residual leaves a suffix of that residual, hence a proper suffix of the same original word. Cancelling a residual from an original word leaves a proper suffix of the original word because the cancelled residual is nonempty. Thus every nonempty reached residual belongs to a fixed finite suffix set.

If the sum of the codeword lengths minus one is M , that suffix set has at most M members; shared suffixes can reduce the actual number. The procedure can add at most those M nonempty residuals and, possibly, the empty residual before stopping. A round can add several residuals, and a final unchanged round certifies closure. The bound concerns the finite reachability construction, rather than the length of every ambiguous event record.

The residual procedure is a precise version of the established Sardinas–Patterson method used in the coding antecedent. Its role here is diagnostic: it either returns a competing decomposition or establishes that the declared unrestricted block vocabulary avoids such ambiguity. Its conclusion remains relative to the codebook, the complete-record convention, and the assumption that each process type has one nonempty event pattern.

6 Derivation of the Decipherability Criterion

This section proves the residual procedure’s decision criterion. Its objective is to establish both directions of the equivalence between a reached empty residual and an ambiguous block history, then explain

termination and the simpler prefix-free case. The proof follows the competing decompositions step by step so that every cancellation has an explicit historical-word interpretation.

Proposition 6.1 (Finite residual criterion). *For a finite set C of distinct nonempty codewords, unrestricted concatenations are uniquely decipherable exactly when the residual construction reaches a fixed set without the empty word. Reaching the empty word supplies two different finite codeword sequences with equal concatenation.*

Proof. For soundness of an ambiguity report, begin with an initial residual r in Equation 5.1. There are distinct codewords u and v with $ur = v$. Regard u and v as the encodings of two one-block histories. Their first types differ because the block assignment is injective. One encoded history extends beyond the other by r .

Maintain the following invariant: a reached residual s has a witness consisting of two histories with different first types, whose encodings agree up to the end of the shorter one and whose unmatched suffix is s . Appending a block to the shorter history preserves the difference between their first types.

Consider the first update case in Equation 5.2. An original codeword u satisfies $ur = s$. Append the type encoded by u to the shorter history. Its newly appended events match the first part of the unmatched suffix. The same encoded history remains longer, with residual r . This produces a witness for the new residual.

In the second update case, $sr = u$. Append the type encoded by u to the shorter history. The appended word first matches the whole existing residual and then extends beyond the previously longer history by r . The orientation reverses, and r again has a witness. If both words have equal length and agree, $r = \epsilon$ and the two encodings coincide.

Induction on the sequence of residual updates therefore supplies a witness for every reached residual. If the empty residual is reached, the witness histories still have different first types and are consequently distinct. Their event encodings agree exactly. The reported ambiguity is thus a genuine ambiguity of the declared finite codebook.

For completeness, suppose two different finite block histories have the same event encoding. Remove their common initial block types. Because every codeword is nonempty, one remaining history cannot be empty while the other is nonempty: their remaining encodings would then have different lengths. Both remaining histories therefore begin with different types.

Their first codewords are prefixes of the same remaining event word. Hence one is a prefix of the other. They cannot be equal because the assignment is injective and their types differ. The unmatched suffix is therefore a nonempty initial residual in Equation 5.1.

Advance the history whose currently consumed encoding is shorter by one of its next original blocks. Such a next block exists unless both consumed encodings already agree at a common endpoint. Both consumed encodings remain prefixes of the same complete event word, so they remain prefix-comparable. Cancelling the shorter matching part produces exactly one of the two residual updates. Each advance consumes one further block from the fixed pair of finite histories.

The process either reaches an earlier common endpoint, which already gives an empty residual between histories with different first types, or reaches the common endpoint of the full equal encodings. Finiteness of the histories guarantees that this happens after finitely many advances. Thus every ambiguity causes the residual construction to reach the empty word.

Finally, the finite suffix argument from the construction bounds all nonempty reachable residuals. The cumulative sequence must therefore either reach the empty word or stabilize. If it stabilizes without the empty word, completeness excludes every ambiguous pair. If an ambiguity exists, soundness and completeness identify it through an empty residual before a false unique-decipherability conclusion can occur. $\square$

The proof distinguishes finite computation from enumeration of all possible histories. The set of unrestricted process histories is infinite whenever the vocabulary is nonempty. The decision procedure remains finite because unfinished disagreements between competing encodings fall into a finite set of

suffix states. The suffix state summarizes precisely the information needed to continue the comparison under the fixed codebook.

The proof also explains the role of orientation. The decision set can omit which history is currently longer because both cancellation directions are included. An ambiguity witness should retain that orientation if it is to reconstruct the actual pair of block sequences. Failing to distinguish the decision state from the witness record could produce an uninterpretable report even when the yes-or-no result is correct.

A simpler sufficient condition is prefix freedom. A codebook is prefix-free when no codeword is a strict prefix of another. Under this condition, the initial residual set is empty and the residual construction immediately stabilizes. The same conclusion has a direct proof that is useful for interpreting explicit boundary designs.

Proposition 6.2 (Prefix-free block recovery). *A finite prefix-free set of nonempty codewords is uniquely decipherable. Its first block can be identified when that codeword has been read, under the assumption that the incoming sequence follows the declared codebook from a block boundary.*

Proof. Suppose two codeword sequences have the same concatenation. Their first codewords are prefixes of the same event word, so the shorter is a prefix of the longer. Prefix freedom forces them to be equal. Cancel the equal first codewords and repeat the argument. Because the sequences are finite and codewords are nonempty, the cancellation ends with both sequences empty. They are the same codeword sequence.

For immediate first-block identification, once the events read match a codeword, prefix freedom excludes any longer codeword beginning with that same word. Under the assumed valid boundary start and codebook membership, the completed codeword therefore determines the first block. $\square$

Prefix freedom is stronger than finite unique decipherability. A code can require additional events, or the end of the record, to determine an earlier block while still assigning one decomposition to every completed finite encoded word. The record-closure section supplies an explicit example. The distinction matters when an institution wants to act before a process record is complete: a theorem about completed records requires a separate argument before it supports an online boundary decision.

The residual result remains an established coding criterion applied to a new interpretive setting. Its proof establishes a model consequence. It leaves the adequacy of the process vocabulary, the authenticity of the event record, and the practical significance of the recovered boundaries to the subsequent analysis.

7 A Document-Handling Dossier and Recoverable Queries

This section develops a fictional workflow in which boundary ambiguity coexists with exact recovery of a coarser query. Its objective is to make the residual witness, competing routine assignments, and process-count calculation explicit. The discussion specifies the vocabulary, follows the ambiguity through the residual construction, and then identifies the assumptions supporting each historical answer.

Consider a document-handling journal with two event types. An event of type a records the writing of an entry, and an event of type b records an inspection of a document. The event positions are assumed to be correctly ordered and individually identified. An entry may precede or follow an inspection depending on the routine being performed. The example stipulates that the required documents and records are available; it makes no empirical claim about an actual office.

The routine vocabulary contains three types. Type A is a standalone entry, with codeword a . Type B is an entry-led inspection, with codeword ab . Type C is an inspection followed by an entry, with codeword ba . The names identify the declared block convention. The event alphabet alone leaves open whether an entry and inspection form one routine or belong to neighboring routines.

Suppose the complete event record is $w = aba$. Under unrestricted concatenation, the two histories shown in Table 1 both reproduce the record. There are no additional decodings. If the first block is A , the remaining suffix ba must be C . If the first block is B , the remaining suffix a must be A . A first

Table 1: Competing process assignments and query answers for the fictional event record aba

Record or query	First admissible history	Second admissible history
Process-type sequence	A, C	B, A
Event segmentation	$a \mid ba$	$ab \mid a$
Internal boundary	After event position 1	After event position 2
Routine containing inspection b	Inspection followed by entry	Entry-led inspection
Total process blocks	2	2
Standalone-entry blocks	1	1

block of type C cannot match the record’s initial event. This exhausts the possible first blocks and their continuations.

The residual construction detects the same ambiguity. Since a is a strict prefix of ab , the initial residual set contains b . Comparing this residual with the original codeword ba leaves a on the opposite side. The reached residual a is itself a codeword, so appending that block reaches the empty residual. The witness is the equality between $a \cdot ba$ and $ab \cdot a$.

The initial comparison can be read as a partial disagreement between histories A and B . The second history has encoded one additional event, b . Appending C to the first history produces aba , extending beyond B ’s encoding ab by the final a . Appending A to the second history then produces the same complete word. The two histories retain different first types throughout, so the empty residual represents a substantive ambiguity.

Attaching distinct identifiers to the three event positions leaves this ambiguity intact if the identifiers carry no process-membership information. Both histories can refer to the same first, second, and third recorded occurrences. Exact timestamps likewise preserve the ambiguity when the block convention depends only on order and permits both patterns at those times. Such fields can improve event identification while leaving process grouping unresolved.

Several queries nevertheless have exact answers within the model. Both decodings contain two process blocks and one standalone-entry block. Both contain one inspection event. They disagree about the first and last process types, the position of the internal boundary, and the type of routine to which the inspection belongs. A report should therefore distinguish the stable answers from the unresolved ones rather than treating the record as either wholly sufficient or wholly useless.

The total process-count result extends beyond this one word. Every codeword in $\{a, ab, ba\}$ contains exactly one event of type a . Let $N_a(w)$ count the occurrences of a in an event word. Concatenation adds these counts, so a history with s process blocks always satisfies $N_a(c^*(h)) = s$. The proof uses the contribution of each block separately: each contributes one a , and flattening concatenates the contributions without deleting or duplicating events.

Proposition 7.1 (Process-count recovery in the dossier). *For every event word admitted by the dossier’s codebook, all admissible process-type decodings have the same number of blocks, equal to the number of a events in the word. Boundary and type recovery can remain ambiguous.*

Proof. Take any decoding $h = k_1 \cdots k_s$. Each $c(k_i)$ contains one a , so additivity of symbol count gives $N_a(c^*(h)) = s$. Since every decoding of w has the same flattened word, each has block count $N_a(w)$. The two decodings of aba demonstrate the simultaneous failure of boundary and type uniqueness. $\square$

The proposition depends on the vocabulary. Adding a routine encoded by aba would introduce a one-block decoding of the same word that previously required two blocks. Allowing a silent routine would permit further count changes without changing the record. The count invariant therefore reflects a specified relation between process blocks and recorded entry events. It should be retained with that assumption rather than generalized into a rule that one journal entry always identifies one actual process.

The distinction can affect a proposed audit. If an inquiry asks how many modeled routines occurred in a faithful closed record, the count may be determined even while their assignments remain disputed.

If an inquiry asks whether the inspection belonged to the first routine or initiated a later one, the same record leaves alternatives. If responsibility depends on a person's role within a particular routine, a further normative and evidential bridge is required. The event word alone supplies neither the relevant role assignment nor its moral or legal significance.

A restrictive workflow language can select one history. For example, a justified condition excluding entry-led inspections followed immediately by standalone entries would remove B, A . That condition should be recorded as part of the model and supported by the evidence appropriate to the inquiry. If the audit is examining whether that very order restriction was violated, presuming it would exclude a candidate violation before the evidence had been assessed.

The dossier therefore demonstrates a useful intermediate result. Process grouping can remain unresolved while selected quantities are exactly determined. The exact quantities retain their modeling premises, and unresolved boundaries retain their alternative presentations. This combination preserves information without converting a convenient reconstruction into an established historical claim.

8 Closed Records and Delayed Boundary Identification

This section examines the temporal conditions of decoding. Its objective is to distinguish uniqueness for a completed finite event word from the ability to identify an earlier block while the record is still arriving. The discussion uses a classical finite code example, derives its residual result, and then constructs an infinite continuation that exposes the need for a closure convention.

Consider the codebook $C = \{0, 01, 11\}$. The word 0 is a strict prefix of 01, so the code is not prefix-free. Its initial residual is 1. Comparing 1 with 11 leaves 1 again, and comparison with the other codewords produces no new residual. The residual set therefore stabilizes at $\{1\}$ without reaching the empty word. The code is uniquely decipherable for finite completed encodings.

The result can also be seen directly for words beginning with one 0 followed by a finite run of 1 events. If the run contains an even number of 1s, the first block is 0 and the remaining events form 11 pairs. If the run contains an odd number, the first block is 01 and the remainder again forms 11 pairs. The final length of the run determines which first block was used.

Before the end of the run is known, that first block can remain unresolved. Any observed prefix consisting of 0 followed by finitely many 1s admits a completed extension with even total parity and a completed extension with odd total parity. The two extensions have different first codewords. Reading more of the same event type therefore need not settle the first boundary at any fixed finite delay.

Example 8.1 (Unique finite decoding with unresolved continuation). The infinite event sequence beginning with 0 and continuing with 1 forever has the decompositions $0 \mid 11 \mid 11 \mid \dots$ and $01 \mid 11 \mid 11 \mid \dots$. Every finite completed encoding under the same codebook has at most one decomposition, while the infinite sequence supports these two different first-block assignments.

The example changes the domain of the uniqueness question. The finite residual theorem concerns finite codeword sequences with a common finite endpoint. The two infinite decompositions never reach a common later block boundary after their initial disagreement. Their residual remains 1 indefinitely. This is consistent with the finite theorem because its completeness proof relies on a shared finite endpoint at which the residual must become empty.

The distinction matters for a record that is merely paused. A journal ending temporarily in 011 has a unique decoding if it is known to be a complete finite encoding. A further 1 changes the completed first-block assignment. Treating the current end of an observed file as the end of the underlying process would therefore introduce a closure premise that the event sequence itself may not establish.

Closure can be supplied in several ways. A trusted protocol may declare a transaction record complete. A separate end marker may delimit a block or a whole record. An independently documented event may establish that the relevant process has ended. A fixed observation window can identify the end of data collection while leaving the underlying process unfinished. These conditions should be distinguished because they support different decoding domains.

The code model assumes that the record begins at a block boundary as well. Starting midway through a routine can produce a suffix that looks like a complete encoding of different routines. A known starting point and a known ending point are therefore part of the record convention used by exact finite decoding. A complete file can still represent only a partial interval of the process being studied.

The example also separates bounded delay from eventual evidence. In a particular finite case, an end marker can make the first block knowable. A general promise to decide after a fixed number of additional events requires a stronger property of the codebook or recording protocol. Prefix freedom supplies one such simple condition for identifying the first block at its completion. Other bounded-delay conditions remain possible, but their proof belongs to a different claim from finite unique decipherability.

For historical interpretation, the conclusion is modest. A decoder should state whether it assumes a closed record, a known block-aligned window, or an ongoing stream. When closure is uncertain, the record can support a set of possible partial and complete histories. Preserving that uncertainty prevents a temporary interruption in observation from becoming an asserted historical boundary.

9 Recording Boundaries and Constrained Histories

This section examines ways to retain the grouping information that a flat event word omits. Its objective is to distinguish mathematical repairs of the record from the evidence needed to justify their historical interpretation. The discussion considers delimiters, process identifiers, admissible-history constraints, and query-specific retention under limited information requirements.

A boundary marker offers a simple formal repair. Choose a symbol # outside the event alphabet and encode each block by its event word followed by that marker. A complete encoded record can then be split at the markers. If the original block assignment is injective, each marked segment determines one process type. The proof uses the marker's exclusion from ordinary event words: every occurrence of # has the declared separator role, and the segments between successive markers are fixed by the record.

This repair depends on a faithful recording convention. A missing marker can merge neighboring blocks, an inserted marker can split one block, and a truncated file can omit the final marker. A technical parser can detect some inconsistencies, but the delimiter itself supplies no general guarantee that the segmentation corresponds to actual activity. The recording procedure must establish what causes a marker to be written and how errors or later corrections are represented.

Markers also leave type collisions unresolved. If two named process types have the same event word, a segment matching that word identifies its boundaries without selecting the type. A type identifier or another discriminating record can repair this difference. The identifier then functions as additional information, and its relation to the activity requires provenance. Merely assigning a distinct database key does not establish that the corresponding process type was correctly identified.

An occurrence identifier has a related but different role. It can connect records asserted to concern the same process instance, including entries held in several places. The identifier can be useful even when a process has varied event patterns. Yet its use presupposes or asserts a co-reference relation. An inquiry into mistaken merging or splitting of cases should preserve the evidence for that relation and permit revisions that remain traceable to earlier assignments.

The mathematical repair should therefore be accompanied by a record of the boundary criterion. A procedure might place a boundary at a documented request, at a completion declaration, at a transfer of responsibility, or at a new case identifier. These conventions answer different questions. The record should state which criterion was used, which evidence supported it, and whether the boundary was contemporaneous or reconstructed later.

Admissible-history constraints provide another route to recovery. A formal protocol can restrict which block sequences are possible, making the flattening map injective on that restricted language even when the unrestricted codebook is ambiguous. The relevant result concerns the pair of codebook and language. Its application requires evidence that the actual history lies within that language, or a clear statement that the reconstruction is conditional on compliance.

A conformance inquiry requires particular care. If a record admits both a compliant and a noncompliant decomposition, choosing the compliant one by restricting the language to permitted histories would settle the contested question by assumption. A more informative analysis retains both histories, identifies the evidence that would distinguish them, and separates conclusions valid under either history from conclusions that depend on the disputed assignment.

Record design can also focus on a narrower target than full reconstruction. In the dossier, a faithful count of a events suffices for total modeled block count under the fixed vocabulary. It leaves the sequence, timing, and process assignments open. If the authorized task concerns only that count, preserving every detail may exceed the information the task needs. A broader audit or correction obligation can justify retaining more, but the additional purpose should be stated rather than inferred from a general preference for complete records.

The record-design problem consequently has several dimensions. Technical uniqueness concerns whether the stored representation has one decoding. Historical warrant concerns whether the representation accurately records the relevant activity. Query sufficiency concerns which answers survive compression. Institutional justification concerns which records should be created, retained, shared, or corrected. A useful design addresses these dimensions separately and records the assumptions that connect them.

The present analysis proposes no universal logging format. It supplies exact examples and proof obligations that a particular format can be tested against. An implementation can use explicit boundary fields, process-type assertions, occurrence identifiers, source links, and revision records in different combinations. The appropriate combination depends on the inquiry, the activity's structure, and the interests of the people whose conduct or experiences the record represents.

10 Process Unity and the Interpretation of Recorded Boundaries

This section examines what the recovered blocks mean beyond their formal decomposition. Its objective is to distinguish procedural grouping from causal, purposive, experiential, and normative forms of continuity. The discussion develops a layered account of process individuation and states a conditional argument for making consequential boundary conventions open to challenge.

A codebook supplies a criterion for grouping event words into named types. Once that criterion is fixed, the recovery problem can be exact. The criterion's interpretation remains a separate matter. A sequence can form one named administrative routine because a protocol treats its events together, while a participant experiences the same sequence as one phase of a much longer undertaking. An explanatory account can likewise connect several routines through a continuing causal dependence.

These criteria need not compete for a single universal boundary. A procedural question asks which operations count as one routine under a specified rule. A causal question asks which dependencies explain an outcome. A purposive question asks which activities contribute to an undertaking and how its aims change. An experiential question asks how the activity is integrated or interrupted for a participant. A responsibility question asks which relations and reasons support continuing duties or attribution. Their answers can use different groupings without logical inconsistency, provided the criterion remains explicit.

The philosophical comparison with Dewey helps identify this plurality. His discussion of integrated experience gives unity a qualitative and developmental role, while the codebook gives it a stipulated procedural role. A decoded block can therefore remain experientially fragmented, and several blocks can participate in one experienced course. The comparison offers a question about the adequacy of a boundary for an inquiry; it supplies no requirement that every institutional process possess experiential fulfillment.

Definition 10.1 (Boundary criterion). A boundary criterion is a stated rule or interpretive basis for treating an interval or connected collection of activity as one process for a specified inquiry. Its application should identify the relevant events, the grounds of their grouping, and the treatment of interruption, continuation, and revision.

The definition leaves several epistemic statuses available. A contemporaneous participant declaration can be an observation or source assertion. An analyst's later grouping can be a proposed interpretation. A protocol can supply a normative classification. A proved decoding result can establish uniqueness conditional on a codebook. The same boundary can acquire support from several sources, but their agreement should be assessed through their actual dependencies. Repeating one administrative assignment across several copied records supplies less independent support than several distinct grounds.

Process identity also differs from process-type identity. The model's *A* and *B* labels classify routine patterns. Repeated occurrences of *A* remain different positions in a history. An occurrence identifier can track a continuing instance through several records, but that identifier's continuity requires a criterion and evidence. A later correction may show that one recorded instance combined two actual undertakings, or that two recorded instances belonged to one continuing undertaking. The record should preserve the earlier assignment while making the refinement visible.

The distinction becomes consequential when boundaries govern evaluation. Suppose an institution records an unresolved request as closed and treats a later communication as a new request. Under that convention, each recorded episode can have a short duration. A participant may maintain that the underlying matter remained unresolved across the boundary. The example is hypothetical. It illustrates how a disagreement about process continuity can affect the interpretation of duration, completion, and repeated effort without requiring disagreement about the recorded communications themselves.

A decoding theorem can determine which segmentation follows from the adopted convention. It cannot establish that the convention answers the participant's question. That judgment requires examining the purpose of closure, the condition the institution claims to have completed, and the reasons for treating later activity as a new undertaking. The event record and the boundary convention should therefore remain separately available for criticism.

Claim 10.2 (Conditional contestability of process boundaries). When an institutional boundary convention materially affects an evaluation or burden, and affected participants have standing to contest the grounds of that effect, a technically unambiguous record leaves an additional obligation to explain and justify the boundary criterion.

The argument begins from the stated normative standing. The effect depends partly on how activity is grouped. Technical uniqueness establishes the consequence of a particular grouping convention, while the participant's standing permits questioning the convention's grounds. An institution must therefore address those grounds in addition to demonstrating that its records were parsed consistently. The conclusion is conditional on the standing and material-effect premises; the coding theorem carries none of that normative force by itself.

This argument permits stable conventions. A shared rule can make coordination, comparison, and correction possible. Stability becomes more defensible when the convention states its scope and records meaningful exceptions or disagreements. A purely discretionary boundary imposed retrospectively to secure a desired evaluation would require a different justification from a stable rule applied transparently with an available correction route.

The analysis also preserves the distinction between process unity and value. A coordinated course of activity can produce harm. An interrupted or fragmented course can contain valuable acts. A record's coherent decomposition therefore supplies limited information about the quality of the activity or the responsibilities arising from it. The proposed relational inquiry asks which dependencies and histories matter to the judgment, while requiring explicit premises for any further normative conclusion.

Historical interpretation benefits from the same discipline. A later narrative can organize dispersed events into an intelligible process, and that organization can generate a useful hypothesis. Its plausibility should be assessed against the traces, alternatives, and reasons that support the grouping. The reader should be able to distinguish recorded boundaries from reconstructed ones, and a reconstruction from a merely illustrative arrangement. That distinction preserves the value of interpretation while keeping its evidential status visible.

11 Objections and Extensions of the Block Representation

This section tests the scope of the finite block model. Its objective is to identify which departures require a revised formal construction and which threaten the historical interpretation of a result. The discussion considers variable realizations, overlapping processes, uncertain event records, empty blocks, vocabulary revision, and the relation between a useful reconstruction and actual occurrence.

The first objection is that actual process types can have several event realizations. A document review may take different branches, repeat a step, or stop after an exception. A single word $c(k)$ then represents a narrow pattern rather than the whole type. One extension assigns a language of event words to each type. The resulting recovery question must distinguish selection of a type from selection of one of its realizations, and the finite residual theorem stated here no longer applies unchanged.

A finite collection of variants can sometimes be treated as a larger vocabulary with explicit variant labels. This transformation preserves a finite codebook but changes the recovery target. Two variants may belong to the same higher-level process type, so variant ambiguity can coexist with type recovery. Conversely, different higher-level types may share the same variant word. An analysis should state the map from variants to types and test the intended query on the resulting decoding set.

The second objection concerns overlapping or interleaved processes. A single event can participate in several undertakings, and the events of one undertaking can be separated by events of another. The present segmentation into contiguous nonoverlapping blocks excludes those structures. A suitable extension would represent a relation between event occurrences and process occurrences, with explicit conditions on order, membership, and shared events. Object-centric approaches provide relevant antecedents, but their formal assumptions should be examined directly before a particular extension is adopted.

Interleaving also changes the interpretation of flattening. A linear event order can be one observed ordering of partially ordered activity. Several processes may share resources or synchronize at particular events. Reconstructing their decomposition can require causal or organizational relations that the event-type string omits. The current codebook example demonstrates one form of grouping loss; it supplies no general decoder for concurrent activity.

The third objection concerns event uncertainty. The formal record assumes correctly identified positions and event types. Actual sources may contain duplicate reports, uncertain timestamps, missing entries, or incompatible descriptions of one occurrence. Those uncertainties should be represented before exact process decoding is interpreted historically. One approach is to construct a family of admissible event records and evaluate the process queries across that family as well as across each record's possible decompositions.

The resulting uncertainty has several layers. A query can be invariant across all decompositions of one candidate event word and still vary across different candidate event words. It can be invariant across recorded events while remaining sensitive to possible unrecorded activity. A report should therefore identify the family over which an answer is stable. Describing an answer as uniquely determined without that domain can overstate its evidential reach.

The fourth objection concerns empty or invisible processes. If a process can occur without any event from the selected alphabet, event-word recovery cannot establish how often it occurred. The model's nonempty-word assumption excludes that possibility. In an application, the exclusion requires a substantive recording claim: every process of the relevant kind must leave at least one included event. A logging policy that merely intends to record such activity supplies a weaker premise than evidence of its actual coverage.

The fifth objection concerns changing process vocabularies. A routine can be subdivided, merged with another, or reinterpreted after a policy change. The same event word can then receive different valid decompositions under different vocabularies. This is a change of model and target, rather than necessarily an error in either decoding. Comparing the results requires a map between the vocabularies and an account of which historical or evaluative question is being preserved.

An evolving vocabulary can also expose a tension in retrospective counts. The dossier's count invariant depends on one entry event per block. Adding a longer block type can change the modeled count

without changing the event record. A time series of process counts therefore requires a stable definition or a documented conversion between definitions. Formal consistency within each period leaves the comparison across periods open until that conversion is justified.

The sixth objection is that a formally ambiguous history can still have an obvious practical interpretation. Rich contextual knowledge may indeed distinguish the alternatives. The appropriate response is to represent the relevant context or state clearly which evidence supplies the distinction. The codebook analysis does not prohibit contextual judgment. It identifies a limit of the flattened record alone and invites the analyst to state the additional grounds that overcome it.

The converse objection is equally important: a formally unique history can be historically false. A fabricated log can match one codeword sequence perfectly. A mistaken vocabulary can exclude the actual process. A closure assertion can conceal continuing activity. The mathematical criterion establishes a conditional relation among records and histories; it should be used within an evidence practice that can challenge the records and the model themselves.

These objections preserve the limited contribution. The finite block representation provides a transparent setting in which boundary loss, query-specific recovery, and record closure can be derived explicitly. Broader representations should retain those distinctions while adding the structures their domains require. The model is useful to the extent that it clarifies an inquiry and its alternatives, with its fit to actual activity remaining open to examination.

12 Open Questions and Evaluation Design

This section identifies the mathematical and empirical work needed to develop the proposal. Its objective is to separate verification of the finite construction from assessment of its usefulness for actual records. The discussion proposes independent algorithmic checks, a comparative annotation study, and open questions concerning process-language constraints, ongoing records, and competing boundary criteria.

Open Problem 12.1 (Recovery under constrained process languages). Develop an inspectable recovery procedure for a specified process-language restriction that returns ambiguity witnesses together with the constraints that exclude or preserve them. The procedure should distinguish descriptive evidence of admissibility from normative requirements being tested for conformance.

The unrestricted residual theorem supplies a useful baseline because its negative result comes with a finite witness. A constrained procedure should make clear whether that witness is excluded by an initial-state condition, an order rule, an observed fact, or an assumption about compliance. The distinction would help an analyst identify the premise carrying a unique reconstruction and assess whether that premise is appropriate to the inquiry.

Open Problem 12.2 (Closure-sensitive partial-history inference). Represent the possible process histories compatible with an ongoing or partly observed record, including uncertainty about its initial boundary and termination. Identify conditions under which a past boundary becomes stable under every permitted continuation.

The infinite example shows that finite unique decipherability alone provides insufficient grounds for a fixed-delay commitment to the first block. A richer analysis could distinguish explicit termination, bounded delays, trusted closure events, and observational silence. Its output should identify boundaries that are stable across permitted continuations separately from boundaries selected by a default stopping rule.

Open Problem 12.3 (Comparison of process-individuation criteria). Construct a method for comparing procedural, purposive, experiential, and responsibility-based groupings of the same recorded activity. The method should preserve their different targets and identify the evidence and normative premises needed to use one grouping in another inquiry.

A mathematical verification programme can begin with finite codebooks over a small alphabet. Enumerate codebooks with bounded word length, apply the residual construction, and compare its result with an independent finite-state search for two distinct accepting parse paths carrying the same event word. The independent search should represent partial positions inside original codewords and a flag recording whether the two parses have diverged. Reaching two block boundaries with that flag set supplies an ambiguity witness.

This comparison tests the residual update against a construction that consumes events directly. It should include prefix-free codes, uniquely decipherable codes with delayed decisions, and ambiguous codes. Empty codewords and duplicate type assignments should be tested as excluded or separately diagnosed conditions. The dossier’s alternative parses and count invariant should receive direct enumeration on bounded records, while the general count proof remains the source of the unrestricted conclusion.

An empirical study would require a bounded activity and a record whose provenance can be examined. The study could compare interpretations based on an event-only record, a record with procedural boundary markers, and a record with boundary assertions linked to their sources and revision history. Participants should have an opportunity to identify missing context and to distinguish an event-report disagreement from a grouping disagreement.

Evaluation should assess several outputs separately. These include agreement about event identity, agreement about process boundaries, agreement about process types, stability of selected counts, recognition of incomplete records, and the ability to locate evidence supporting a disputed boundary. A single agreement score can conceal the fact that participants agree on a count while disagreeing about the activity to which responsibility is attributed.

The study should also vary the inquiry. A record adequate for a throughput count may be inadequate for explaining repeated effort or assessing whether a matter remained unresolved. Asking participants to answer several declared queries from the same record would help identify the relationship between record structure and task-specific sufficiency. Such a study would test the usefulness of the distinctions without assuming that one segmentation is appropriate for every purpose.

Boundary conventions deserve direct scrutiny when they affect evaluations. The study could present alternative defensible conventions, document their consequences, and examine how participants assess their reasons. This component would require an ethical design appropriate to the people and records involved. It should preserve participants’ disagreement about the task or category as substantive material, rather than treating it automatically as annotation error.

The present paper reports analytic constructions and proposed evaluations. Actual error rates, annotation behavior, institutional effects, and the comparative benefits of particular logging formats remain open. Further literature work should examine more general code languages, event-log uncertainty, concurrent process representations, and philosophical accounts of process individuation. Each source should be verified before being used to extend the manuscript’s claims.

13 Conclusion

This section states the completed argument and its limits. Its objective is to connect the process-boundary problem, the finite decoding results, and their historical interpretation. The conclusion identifies the recoverable distinctions, the role of record closure, and the further grounds required for process and responsibility judgments.

Process-history information can be lost before an event word is compressed into a composite mechanism or endpoint. Removing the boundaries between named routines can leave the complete event sequence unchanged while merging distinct process histories. A record’s sufficiency should therefore be assessed against a declared target, including the difference between event order, segmentation, process type, and a coarser historical query.

For a finite vocabulary of nonempty fixed event patterns, exact process-type recovery is injectivity of the flattening map on the admissible histories. In the unrestricted case with distinct codewords, the classical residual construction gives a finite test. Its proof follows competing histories through prefix

cancellation and uses a finite suffix carrier to establish termination. The paper reconstructs that result explicitly and applies it without a claim of mathematical priority.

The fictional document-handling dossier shows that several process decompositions can share an exact count. Its event word *aba* admits two boundary patterns, while every admitted history has one block per *a* event under the fixed vocabulary. The stable count and the unresolved boundary should be reported together. Changing the vocabulary or questioning record coverage changes the premises of that conclusion.

The closure example establishes a different limit. A code can be uniquely decipherable for finite completed records while leaving an early boundary unresolved under arbitrarily long continuation. An observed pause, the end of a data file, and the documented completion of a process consequently support different inferences. A decoder should retain the record convention on which its uniqueness claim depends.

Boundary markers, process identifiers, and language constraints can improve recoverability, but their historical interpretation requires provenance and evidence. Procedural, causal, purposive, experiential, and normative groupings can answer different questions about the same activity. Technical consistency within one convention leaves the justification of that convention open when it carries consequential institutional effects.

The paper offers these distinctions for further discussion. Its finite results provide exact consequences within a declared representation. The adequacy of the representation, the truth of a recovered history, and the responsibilities associated with a process remain matters for additional evidence and argument.

References

- [1] John Dewey. Having an experience. Chapter III of *Art as Experience*, 1934. Chapter text hosted by Western Oregon University:
https://people.wou.edu/~girodm/670/having_an_experience.pdf.
- [2] Robert Sedgewick and Kevin Wayne. Data compression. *Algorithms*, Fourth Edition booksite, Section 5.5. <https://algs4.cs.princeton.edu/55compression/>.
- [3] Wil van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *WIREs Data Mining and Knowledge Discovery*, 2:182–192, 2012.
- [4] Wil M. P. van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In *Software Engineering and Formal Methods: SEFM 2019*, pages 3–25. Springer, 2019.
- [5] World Wide Web Consortium. PROV-DM: The PROV data model. W3C Recommendation, 30 April 2013, 2013. <https://www.w3.org/TR/2013/REC-prov-dm-20130430/>.