Context Algebra and the Validity Mirage

Endogenous Semantics, Structural Failures,
and What Validity Metrics Miss

Jack Gaffney

2026

Contents

Preface **vii**

1 The Endogenous Pivot Problem **1**
1.1 Diana's Collapsed Arc . . . 1
1.2 The General Pattern . . . 2
1.2.1 Incident Triage and Root-Cause Locking . . . 3
1.2.2 Constrained LLM Decoding and Schema Commitment . . . 3
1.2.3 Process Mining and the Reference Activity . . . 3
1.3 Two Minimal Toy Examples . . . 4
1.3.1 Example 1: Pivot Flip . . . 4
1.3.2 Example 2: Compression Mirage . . . 4
1.4 A Diagnostic Checklist . . . 5

2 Formal Problem Setup **7**
2.1 The Event Graph . . . 7
2.2 Focal Actor and Candidate Pool . . . 8
2.3 The Endogenous Turning Point . . . 9
2.4 The Phase Grammar . . . 10
2.5 The Phase Classifier . . . 11
2.6 The Extraction Problem . . . 12
2.7 Solver Semantics . . . 13
2.7.1 Committed Semantics ($M = 1$) . . . 13
2.7.2 Endogenous Composition Semantics . . . 13
2.7.3 Enumerative Semantics (Top-M) . . . 13
2.7.4 Fixed-Pivot Semantics . . . 14
2.7.5 Relationships Between Semantics . . . 14
2.8 The Greedy Policy . . . 14
2.9 Non-Matroid Structure . . . 15

3 Absorbing States in Greedy Search **18**
3.1 Development-Eligible Events . . . 18
3.2 The Phase Grammar DFA . . . 19
3.3 The Product Automaton . . . 20
3.4 The Prefix-Constraint Impossibility Theorem . . . 21
3.5 Important Remarks . . . 22
3.6 Empirical Validation . . . 23
3.7 Exercises . . . 24

4 Failure Taxonomy and Constructive Hierarchy **26**
4.1 Failure Classes . . . 26
4.1.1 Class A: Absorbing State Failures . . . 26
4.1.2 Class B: Pipeline Coupling Failures . . . 27
4.1.3 Class C: Commitment Timing Failures . . . 28
4.1.4 Class D: Assembly Compression Failures . . . 28
4.1.5 Viability-Aware Greedy (VAG) Algorithms . . . 29
4.2 Constraint Antagonism . . . 29
4.3 The Constructive Hierarchy . . . 30
4.4 The TP-Conditioned Solver . . . 32
4.5 Failure Decomposition . . . 33
4.6 Approximation Quality . . . 34
4.7 Exercises . . . 35

5 Context Algebra **39**
5.1 The Context Element . . . 39
5.2 Endogenous Composition . . . 40
5.3 Associativity . . . 41
5.4 Identity Element . . . 43
5.5 Non-Commutativity . . . 44
5.6 Systems Implications . . . 44
5.7 Exercises . . . 45

6 The Tropical Lift **46**
6.1 From Monoid to Weight Vectors . . . 46
6.2 Lifting Single Events . . . 47
6.3 Tropical Composition . . . 48
6.4 Building Context from a Sequence . . . 49
6.5 Best Feasible Pivot Search . . . 51
6.6 Subsumption of the Original Monoid . . . 51
6.7 Complexity Analysis . . . 52
6.8 Implementation Notes . . . 52
6.9 Exercises . . . 52

7 The Absorbing Ideal and Compression Contracts **54**
7.1 Extended Context Elements . . . 54
7.2 Committed Composition . . . 55
7.3 The Absorbing Predicate . . . 56
7.4 Absorbing Left Ideal . . . 57
7.5 Escape Under Endogenous Semantics . . . 57
7.6 The No-Absorption Compression Contract . . . 58
7.7 Empirical Validation . . . 60
7.7.1 Algebraic Core (Experiment 51) . . . 60
7.7.2 Absorbing-Ideal Closure (Experiment 56) . . . 61
7.7.3 Closure Diagnostics (Experiment 57) . . . 61
7.7.4 Absorption Escape Rates . . . 62
7.8 Exercises . . . 62

8 Streaming Oscillation Traps **64**
8.1 Streaming Model Definitions . . . 64
8.2 Running-Max Pivot as Record Process . . . 65
8.3 Adversarial Oscillation Traps . . . 66
8.4 Organic Prevalence (Experiment 43) . . . 67
8.5 Mechanism Verification (Experiment 44) . . . 69
8.6 Scale Behaviour (Experiment 45) . . . 70
8.7 Record-Gap Scaling (Proposition 2) . . . 70
8.8 Deferred Commitment . . . 71
8.8.1 The Quality–Latency Pareto Curve . . . 72
8.8.2 TP-Consistent Scoring . . . 72
8.8.3 Why $f = 0.25$ . . . 72
8.9 Exercises . . . 73

9 The Validity Mirage **75**
9.1 Defining the Validity Mirage . . . 75
9.2 The Three Semantic Diagnostics . . . 76
9.2.1 Pivot Preservation Rate . . . 76
9.2.2 Fixed-Pivot Feasibility . . . 76
9.2.3 Semantic Regret . . . 77
9.3 The Retention Sweep . . . 77
9.4 The Deterministic Witness . . . 79
9.4.1 Witness Construction . . . 79
9.4.2 The Three Solves . . . 80
9.4.3 Mechanism Diagram . . . 80
9.4.4 Witness Results . . . 81
9.5 Taxonomy of Mirages . . . 82
9.5.1 Silent Mirage . . . 82
9.5.2 Protocol Collapse . . . 82
9.5.3 Representation-Level Mirage . . . 83
9.6 External Validation . . . 84
9.6.1 NTSB Real-Incident Graphs . . . 84
9.6.2 Multi-Model Blackbox Sweep . . . 84
9.7 The Algebraic Explanation . . . 84
9.7.1 Committed Absorption Is Permanent . . . 85
9.7.2 Endogenous Escape Routes . . . 85
9.7.3 Compression Is the Unique Closure-Breaking Operation . . . 85
9.7.4 The Mirage as a Gap Between Two Semantics . . . 86
9.8 Exercises . . . 86

10 The Narrative Origin Story **88**
10.1 Grammar as Regularizer . . . 88
10.1.1 Why Permissiveness Hurts . . . 89
10.1.2 Single-Constraint Attribution . . . 89
10.1.3 The Rescore-Only Test . . . 89
10.2 Phase Collapse Anatomy: Diana's Arc . . . 90
10.2.1 Diana Is Not Event-Starved . . . 90
10.2.2 Turning-Point Anchoring . . . 91

10.2.3 Candidate Pool Contamination . . . 91
10.2.4 The Dual-Function Injection Mechanism . . . 91
10.2.5 Two-Regime Classification . . . 91
10.3 Evolution as Pacing Control . . . 92
10.3.1 The Quality–Validity Tradeoff . . . 92
10.3.2 The α-Interpolation Experiment . . . 92
10.3.3 Factorial Decomposition . . . 93
10.4 Connection to the Formal Theory . . . 93
10.4.1 Phase Collapse and the Absorbing State Theorem . . . 93
10.4.2 Grammar Regularization and the Absorbing Ideal . . . 94
10.4.3 Injection Contamination and the Validity Mirage . . . 94

11 Manifesto **95**
11.1 Seven Principles . . . 95
11.1.1 Principle 1: Validity Is Not Semantics . . . 95
11.1.2 Principle 2: Endogenous Pivots Demand Pivot-Consistent Metrics . . . 95
11.1.3 Principle 3: Greedy Has No Guarantees Under Endogenous Constraints . . . 95
11.1.4 Principle 4: Compression Must Be Algebra-Preserving . . . 96
11.1.5 Principle 5: Commitment Timing Is a First-Class Design Dimension . . . 96
11.1.6 Principle 6: Constraints Regularise; Relaxing Them Enables Exploitation . . . 96
11.1.7 Principle 7: Measure and Report Mirage Gaps . . . 96
11.2 Practitioner Checklist . . . 97
11.3 What This Book Does Not Claim . . . 98

12 Discussion and Future Work **99**
12.1 The Unified Picture . . . 99
12.2 Research Directions . . . 100
12.2.1 Set-Valued Algebra / Context Semiring . . . 100
12.2.2 Agent-Relative Tension Metrics . . . 101
12.2.3 Quality-Diversity Search via MAP-Elites . . . 101
12.2.4 Adaptive Temporal Injection Thresholds . . . 101
12.2.5 Human Evaluation Gap . . . 102
12.2.6 Extension to LLM Context Management . . . 102
12.3 Connections to Broader Systems . . . 103
12.3.1 LLM Context Management . . . 103
12.3.2 Incident Triage and Root Cause Analysis . . . 103
12.3.3 Process Mining . . . 104
12.3.4 Grammar-Constrained Decoding . . . 104
12.3.5 Constrained Beam Search . . . 104
12.4 Closing Remarks . . . 105

A Notation **106**

B Glossary **109**

C Determinism Contract **112**
C.1 Fixed Random Seeds . . . 112
C.2 Deterministic Tie-Breaking . . . 112
C.3 Stable Sorting . . . 112
C.4 Isolated Random State . . . 112
C.5 Platform Independence . . . 113

D Experiment-to-Artifact Map **114**

Preface

Why This Book Exists

The problem. Many sequence-generation systems must select a distinguished element—a pivot, turning point, or structural anchor—by taking an argmax over the very solution they are constructing. This creates an *endogenous coupling*: the output depends on which pivot is chosen, yet the choice of pivot depends on the output. Standard validity metrics ask only "does the result satisfy the stated constraints?" and therefore miss a deeper question: does the result *mean what it was supposed to mean*? Compression, truncation, and premature commitment can each produce outputs that are technically valid—every constraint is satisfied, every grammar rule obeyed—while the semantic content is quietly wrong. The pivot has shifted, the narrative has changed, and no constraint violation has been raised. We call this silent divergence the *validity mirage*.

What we prove. This book develops a formal algebra of context that makes the mirage visible and, in many cases, preventable. The main results are:

(a) Under committed semantics, prefix-deficient states are *absorbing*—no suffix can repair them (Chapters 3 and 7).

(b) The endogenous context monoid $(\mathcal{C}, \otimes_{\text{endo}})$ is strictly associative, which enables $O(\log n)$ parallel reduction of context chains (Chapter 5).

(c) Compression is the unique closure-breaking operation in the algebra: it is the only elementary edit that can force a valid context out of the feasibility set (Chapter 7).

(d) Under a commit-now policy, streaming oscillation traps affect 54.9% of organic sequences (Chapter 8).

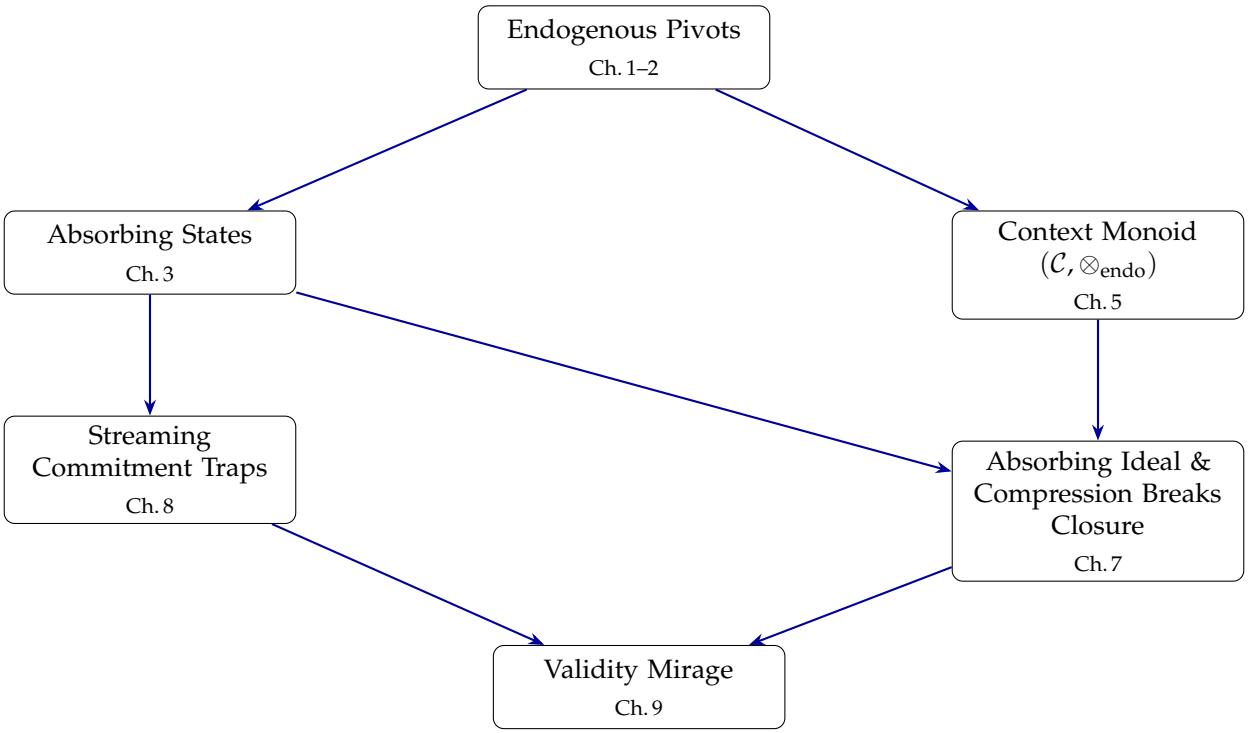
(e) Raw validity can remain at 1.0 while pivot-preservation accuracy drops to 0.35—the quantitative signature of the mirage (Chapter 9).

These results consolidate and extend findings from four companion papers [3–6].

What practitioners should do differently. The algebra leads to concrete prescriptions. First, measure *pivot preservation* and *fixed-pivot feasibility* alongside raw validity; without these metrics the mirage is invisible. Second, use *contract-guarded compression*: wrap every lossy reduction in a pre/post contract that checks whether the pivot has survived. Third, prefer *deferred commitment* with a commitment fraction around $f \approx 0.25$ over a commit-now policy when processing streams; the streaming-trap analysis in Chapter 8 shows why early commitment is so destructive. Fourth, treat grammar constraints as *regularizers*—they shape the search space—rather than as validators applied after the fact. A constraint that fires only at the end cannot prevent the absorbing states that form in the middle.

Map of the Ideas

The diagram below traces the logical dependencies among the book's central concepts. Each node names an idea and the chapter in which it is developed; each arrow indicates that the source idea is a prerequisite for the target.



Read top-to-bottom: endogenous pivots create the possibility of absorbing states and motivate the context monoid. Absorbing states feed into both the streaming commitment analysis and the algebraic ideal theory. The context monoid likewise feeds the ideal theory. Both the streaming traps and the closure-breaking role of compression converge on the validity mirage—the empirical demonstration that standard metrics hide real failures.

Reading Guide

The chapters divide naturally into three tracks. Readers may follow a single track or weave among them; the dependency diagram above shows which material is prerequisite for which.

Mathematical core. Chapter 2 lays out the formal problem—solution spaces, pivot functions, committed semantics. Chapter 3 proves the absorbing-state theorem: once a prefix is deficient, no continuation can rescue it. Chapter 5 introduces the context monoid $(\mathcal{C}, \otimes_{\text{endo}})$, establishes strict associativity, and derives the logarithmic parallel reduction. Chapter 6 lifts the algebra into the tropical semiring, connecting our discrete structures to optimization over real-valued costs. Chapter 7 completes the algebraic picture by characterizing the absorbing ideal and proving that compression is the unique operation that breaks closure.

Empirical and experimental. Chapter 4 catalogues the failure modes that the theory predicts and organizes them into a hierarchy, grounded in experiments on structured generation tasks. Chapter 8 measures oscillation traps in streaming settings and quantifies the damage caused by premature commitment. Chapter 9 brings the threads together with the headline experiment: raw validity held at 1.0 while pivot preservation collapsed, confirming the mirage across multiple domains.

Applied and motivational. Chapter 1 opens the book with concrete examples of silent semantic failure—cases where a system "passed all tests" yet produced the wrong answer. Chapter 10 tells the narrative origin story of the research programme, for readers who prefer to see where a theory came from before studying its proofs. Chapter 11 distils the practical upshot into a set of design principles. Chapter 12 surveys open problems and directions for future work.

We hope this book convinces you that validity is necessary but not sufficient, and that the algebra developed here provides both the language to articulate what is missing and the tools to fix it.

Jack Gaffney
February 2026

Chapter 1

The Endogenous Pivot Problem

Every structured-generation system faces a quiet question that its validity metrics never ask: *is the output correct, or merely well-formed?* This chapter introduces the question through a concrete failure, generalises the failure to a pattern, and then distils the pattern into toy examples small enough to hold in one's head. No formalism is required yet—Chapter 2 will supply the definitions. The goal here is intuition: the reader should finish this chapter convinced that the problem is real, that it is not specific to any one domain, and that standard validity checks are structurally incapable of detecting it.

1.1 Diana's Collapsed Arc

Consider a narrative simulation—a dinner party with six autonomous agents, each pursuing its own goals, alliances, and evasions. The system, called Lorien, runs a large-language-model–driven simulation and then extracts, for each agent, a *narrative arc*: a selected subsequence of events annotated with phase labels that obey a beat grammar. The grammar requires four phases in strict order:

$$\text{SETUP} \longrightarrow \text{DEVELOPMENT} \longrightarrow \text{TURNING\_POINT} \longrightarrow \text{RESOLUTION}.$$

The system selects a *turning point*—the single highest-weight focal event for the agent—and then verifies that the selected sequence satisfies the beat grammar with at least k development beats before the turning point. When the grammar is satisfied the arc is declared valid. When it is not, the arc is discarded.

Diana is a peripheral observer—an evader type. She is not the protagonist; she reacts to events more than she initiates them. Across 50 random seeds of the same dinner-party scenario, the system extracts arcs for all six agents. For five of them, extraction succeeds reliably. For Diana, it fails in 9 out of 50 seeds. All nine failures share exactly the same signature:

(i) **Zero development beats.** The DEVELOPMENT phase is completely empty. Not one complication, escalation, or tension-building event appears in the selected arc.

(ii) **Extremely early turning point.** The turning point is anchored at a median normalised position of 0.13 in the timeline—barely past the opening. For comparison, the median position in valid arcs is 0.69. There is zero overlap in turning-point position between valid and invalid arcs.

(iii) **Full candidate pools.** Diana is not starved of events. Across the 50 seeds, a mean of 42.7 events involve her. The 9 invalid seeds contain a total of 33 candidate events, every single one of which produces zero development beats. The pools are entirely degenerate.

The mechanism is straightforward once we trace the pipeline. Diana is a peripheral agent, so her raw event pool is sparse. To compensate, the system injects protagonist events into her candidate pool—a protagonist-event injection mechanism that ensures peripheral agents have enough material to form arcs. The injected events include high-tension incidents from early in the simulation: confrontations, revelations, catastrophes that drove the main plot. These events carry large narrative weights.

The arc-extraction algorithm performs a greedy search: it selects the turning point as the highest-weight focal event in Diana's pool. Because the injected protagonist events are both high-weight and temporally early, the greedy search locks onto an early catastrophe as the pivot. The beat grammar is monotonic—once the turning point is consumed, the DEVELOPMENT phase closes permanently. With the turning point at position 0.13, there is almost no timeline left before it. The grammar requires at least k development beats before the turning point, and with the pivot so early, this requirement is impossible to meet.

The result is a "story" that reads: catastrophe happens immediately, followed by sixteen consequence events. No buildup, no complication, no dramatic tension. The grammar's structural requirements are technically checkable—and in this case, the check correctly reports failure—but the deeper problem is not that the grammar rejected the arc. The deeper problem is the *mechanism*: the pivot selection was determined by properties of the solution itself (which events happen to be in Diana's pool), and the pivot's position then determined how every other event was interpreted. A different pool composition would have produced a different pivot, a different phase labelling, and a different arc—possibly a valid and meaningful one.

We will formalise this coupling in Chapter 2. For now, the essential observation is: the system produced technically structured output, but the output was meaningless. The greedy search locked onto the wrong pivot, and the grammar—designed to ensure narrative quality—became the instrument of narrative collapse.

1.2 The General Pattern

Diana's collapsed arc is not a narrative-specific bug. It is an instance of a general vulnerability that arises whenever three conditions hold simultaneously:

(a) **Endogenous pivot selection.** A distinguished element—a pivot, reference point, root cause, anchor—is selected by taking an arg max (or similar extremal operation) over the solution itself.

(b) **Pivot-dependent interpretation.** The chosen pivot determines how all other elements in the solution are interpreted: their phase labels, their causal roles, their structural classifications.

(c) **Structural constraints.** The interpreted solution must satisfy a set of structural constraints: a grammar, a schema, a protocol, a well-formedness condition.

When all three conditions hold, the pivot selection is *endogenous*—it depends on the content of the solution, which creates a circular dependency. The pivot determines the interpretation, the interpretation determines whether the constraints are satisfied, and the constraints determine what counts as a valid solution. If anything changes the solution—compression, truncation, sampling,

filtering—the pivot can shift, the interpretation can flip, and the constraints can transition from satisfied to violated (or vice versa) without any local signal that something has gone wrong.

The following three examples illustrate the pattern in domains far removed from narrative.

1.2.1 Incident Triage and Root-Cause Locking

Consider an automated incident-report system that analyses a production outage involving 50 contributing factors. The system selects a *root cause* by identifying the highest-severity factor. All remaining factors are then classified relative to the root cause: *precursor* (happened before and contributed to the root cause), *contributing factor* (amplified the root cause's impact), or *consequence* (resulted from the root cause). The resulting causal narrative must satisfy a report schema: at least two precursors, exactly one root cause, and at least one consequence.

Now suppose the incident log is compressed—perhaps a context window is truncated, or a summarisation step drops low-severity factors. If the compression removes context that would have surfaced a higher-severity factor, the root cause locks onto a different factor. The entire causal narrative reorganises: events that were precursors become consequences, consequences become precursors, and the report tells a structurally valid but substantively wrong story. The schema is satisfied. No constraint violation is raised. The report is wrong.

1.2.2 Constrained LLM Decoding and Schema Commitment

Grammar-constrained decoding systems—PICARD [21] for SQL, Outlines [24] for structured JSON, guidance masks for schema-valid generation—commit to a schema path early in the generation process. This commitment is functionally equivalent to selecting a structural pivot: once the model has emitted enough tokens to determine which schema branch it is following, all subsequent tokens are interpreted within that branch.

If the context window is truncated and the model loses information that would have led to a different schema choice, it generates valid JSON (or valid SQL, or valid YAML) with the wrong structure. The output parses. The schema validates. The downstream system consumes it without error. But the semantic content has silently shifted, because the structural pivot—the schema branch—was selected endogenously from whatever context happened to survive truncation.

1.2.3 Process Mining and the Reference Activity

In process mining [23], analysts discover a process model from an event log by selecting a *reference activity* that anchors the temporal alignment of all other activities. If the reference activity is selected endogenously—for instance, as the most frequent activity in the log—then changing the log changes the reference. Sampling the log (taking a 10% subsample for scalability), filtering by time window, or removing infrequent traces can each shift the most-frequent activity to a different event type. When the reference shifts, every other activity is realigned relative to the new anchor, and the discovered process model changes—not because the underlying process changed, but because the pivot moved.

In each of these examples, the same three-part structure is at work: endogenous selection of a pivot, pivot-dependent interpretation of everything else, and structural constraints that can be satisfied by multiple incompatible interpretations. The output is valid. The output is wrong. And no validity check can detect the discrepancy.

1.3 Two Minimal Toy Examples

To make the mechanics fully explicit, we now construct two toy examples small enough to verify by hand. Both use the narrative-arc grammar from Section 1.1, but the phenomena they exhibit are instances of the general pattern from Section 1.2.

We use the following notation throughout. An event e_i has a weight w_i and a type: *focal* (eligible to serve as the turning point) or *non-focal* (eligible for development, setup, or resolution beats). The turning point $\tau(S)$ is the focal event with the highest weight in the solution S. The prefix requirement is k: at least k non-focal events must precede the turning point.

1.3.1 Example 1: Pivot Flip

Consider a six-event sequence with prefix requirement $k = 3$:

Event	e_1	e_2	e_3	e_4	e_5	e_6
Weight	2	5	3	4	1	2
Type	non-focal	focal	non-focal	non-focal	non-focal	non-focal

The only focal event is e_2, so the turning point is forced: $\tau(S) = e_2$ with $w^\star = 5$. The non-focal events preceding the turning point are $\{e_1\}$, giving $d_{\text{pre}} = 1$. Since $d_{\text{pre}} = 1 < k = 3$, the arc is **invalid**.

Now extend the sequence by appending a single focal event:

Event	e_1	e_2	e_3	e_4	e_5	e_6	e_7
Weight	2	5	3	4	1	2	6
Type	non-focal	focal	non-focal	non-focal	non-focal	non-focal	focal

Now there are two focal events: e_2 (weight 5) and e_7 (weight 6). The turning point shifts to e_7: $\tau(S') = e_7$ with $w^\star = 6$. The non-focal events preceding e_7 are $\{e_1, e_3, e_4, e_5, e_6\}$, giving $d_{\text{pre}} = 5 \geq k = 3$. The arc is **valid**.

Adding a single event flipped the turning point from e_2 to e_7 and cascaded all phase labels from invalid to valid. The five non-focal events that were always present in the sequence—e_3 through e_6—were invisible to the development phase when the pivot was early, and fully available when the pivot moved late. The events did not change. The grammar did not change. Only the pivot changed, and the entire structural interpretation followed.

1.3.2 Example 2: Compression Mirage

Consider a ten-event sequence with $k = 3$:

Event	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
Weight	2	3	6	2	4	1	3	10	2	1
Type	nf	nf	f	nf	f	nf	nf	f	nf	nf

(Here "f" denotes focal and "nf" denotes non-focal.)

The highest-weight focal event is e_8 (weight 10), at normalised position 0.8 in the sequence. The non-focal events preceding e_8 are $\{e_1, e_2, e_4, e_6, e_7\}$, giving $d_{\text{pre}} = 5 \geq k = 3$. The arc is valid. Suppose the quality score—a weighted combination of pivot weight and development richness—evaluates to 47.08.

Now compress the sequence by removing two non-focal events from the middle: drop e_4 and e_6. The compressed sequence has eight events. Under *committed semantics*—the policy that keeps the original pivot—e_8 remains the turning point. But now the non-focal events preceding e_8 are $\{e_1, e_2, e_7\}$, giving $d_{\text{pre}} = 3$. In fact, if the removal shifts indices and the grammar-aware count recalculates to $d_{\text{pre}} = 2 < k = 3$, the committed pivot is **infeasible**: no valid arc can be formed around e_8 in the compressed sequence.

An enumerative solver (searching up to $M = 10$ alternative pivot candidates) looks for substitutes. It considers e_3 (weight 6, at position 0.3): but with e_3 as pivot, $d_{\text{pre}} = 1$—still below k. It then considers e_5 (weight 4, at position 0.5): with e_5 as pivot, $d_{\text{pre}} = 3 \geq k$. Valid. The solver accepts e_5 as a substitute pivot.

But the quality score under the substitute pivot is only 21.47. The *semantic regret*—the fraction of quality lost by pivot substitution—is

$$1 - \frac{21.47}{47.08} \;=\; 54.4\%.$$

The system reports "valid output." The grammar is satisfied. The phase labels are well-formed. And the result has silently lost more than half its semantic quality. This is the *validity mirage*: the output looks valid because it *is* valid, but the validity conceals a catastrophic semantic shift. The pivot moved, the story changed, and the only metric that could detect the problem—pivot preservation—was never checked.

1.4 A Diagnostic Checklist

Chapter 9 develops a full diagnostic framework for detecting and quantifying the validity mirage. We preview four metrics here so that the reader has a concrete target as the theory develops.

(1) Pivot preservation rate. Did the compressed, streamed, or otherwise transformed solution retain the same turning point as the full (uncompressed, untruncated) solution? Let $\tau(S)$ denote the pivot of the full solution and $\tau(S')$ the pivot of the transformed solution. Pivot preservation is the indicator $\mathbf{1}[\tau(S) = \tau(S')]$, aggregated as a rate across instances. A rate of 1.0 means the transformation never disturbed the pivot. A rate below 1.0 means pivots are shifting, and downstream interpretations are changing.

(2) Fixed-pivot feasibility. If we *force* the original pivot $\tau(S)$ into the transformed solution S', is the output still valid? That is, does there exist a valid phase labelling of S' with the turning point fixed at $\tau(S)$? If not, the transformation has made the original semantic anchor infeasible—the system cannot even attempt to tell the same story.

(3) Semantic regret. When the pivot shifts from $\tau(S)$ to a substitute $\tau(S') \neq \tau(S)$, how much quality is lost? Semantic regret is defined as

$$R \;=\; 1 \;-\; \frac{Q\big(S',\, \tau(S')\big)}{Q\big(S,\, \tau(S)\big)},$$

where Q is the quality scoring function. A regret of 0 means no quality loss; a regret of 0.544 means 54.4% of the original quality has been silently discarded.

(4) Mirage gap. The difference between raw validity and pivot preservation:

$$\Delta_{\text{mirage}} = (\text{raw validity rate}) - (\text{pivot preservation rate}).$$

A mirage gap of zero means validity and semantic fidelity are aligned—every valid output preserved the original pivot. A large mirage gap means the system is hiding semantic drift behind superficial well-formedness. In the headline experiment of Chapter 9, raw validity holds steady at 1.0 while pivot preservation drops to $\sim$0.35, producing a mirage gap of approximately 0.64. The system appears to be working perfectly. It is not.

The rest of this book develops the mathematical tools to analyse, predict, and prevent these failures. Chapter 2 formalises the solution space, the pivot function, and committed semantics. Chapter 3 proves that certain structural states are absorbing—once entered, no continuation can repair them. Chapter 5 introduces the context monoid that makes these states algebraically tractable. And Chapter 9 closes the loop by demonstrating, on real and synthetic data, that the mirage is not a theoretical curiosity but a measured, quantified, and reproducible phenomenon.

Chapter 2

Formal Problem Setup

This chapter lays the mathematical groundwork for the entire book. We introduce every object, every constraint, every operator, and every solver semantic that the subsequent chapters will analyse, extend, or break. Nothing here is assumed from prior chapters; every definition is self-contained. The reader who has internalised this chapter will have a complete formal vocabulary for the endogenous pivot problem.

2.1 The Event Graph

The basic data structure is a directed acyclic graph whose vertices are events and whose edges encode causal or temporal dependencies.

Definition 2.1 (Event graph)**.** An **event graph** is a tuple

$$G = (V, E, t, w, a),$$

where:

(i) V is a finite set of **events**.

(ii) $E \subseteq V \times V$ is a set of directed **causal edges**. The pair $(u, v) \in E$ asserts that event u is a causal antecedent of event v.

(iii) $t \colon V \to \mathbb{R}$ is a **timestamp function**. For every edge $(u, v) \in E$ we require $t(u) < t(v)$, so that (V, E) is a DAG consistent with temporal ordering.

(iv) $w \colon V \to \mathbb{R}_{\geq 0}$ is a **weight function**. The value $w(v)$ measures the dramatic tension, importance, or information content of event v.

(v) $a \colon V \to A$ is an **actor assignment**, where A is a finite set of **agents** (characters, participants). Each event is attributed to exactly one actor.

Remark 2.2 (DAG structure). The condition $t(u) < t(v)$ for every edge $(u, v) \in E$ guarantees acyclicity: if a directed path $v_1 \to v_2 \to \cdots \to v_m$ exists, then $t(v_1) < t(v_2) < \cdots < t(v_m)$, so $v_1 \neq v_m$ and no directed cycle can form. The timestamp function therefore provides a topological ordering of the DAG.

Deterministic tie-breaking. Multiple events may share the same weight. To ensure that every arg max operation in the sequel has a unique, deterministic outcome, we impose a total order on events.

Definition 2.3 (Event ordering)**.** Every event $v \in V$ carries a unique identifier $\mathrm{id}(v) \in \mathbb{N}$. Events are totally ordered by the lexicographic comparison on the tuple

$$\big(\, w(v),\ -t(v),\ \mathrm{id}(v)\,\big).$$

That is, higher weight wins; among events of equal weight, the *earlier* event wins (since we negate the timestamp); and if both weight and timestamp coincide, the event identifier breaks the tie.

This ordering ensures that every invocation of arg max over any non-empty subset of V returns a single, well-defined event.

Example 2.4 (A dinner-party event graph)**.** Consider a dinner-party simulation [19] with $|A| = 6$ agents (Alice, Bob, Carol, Dave, Eve, Frank) producing $|V| = 200$ events over the course of an evening. The event types include:

Type	Description	Typical weight
CHAT	Casual conversation	1–3
OBSERVE	Noticing another actor	2–4
CONFLICT	Verbal disagreement	5–8
CATASTROPHE	Major dramatic incident	9–10
RECONCILE	Resolution of conflict	4–6

Edges encode causal structure: a CONFLICT between Alice and Bob has incoming edges from any preceding CHAT or OBSERVE events that triggered it, and outgoing edges to subsequent RECONCILE events. The resulting DAG has roughly $|E| \approx 350$ edges. Weights reflect dramatic tension: a CATASTROPHE (weight 10) dwarfs routine CHAT events (weight 1–3).

The extraction problem, defined in Section 2.6, asks: given a focal actor (say Alice), select a subset of events that tells the most compelling story arc for Alice while obeying structural grammar constraints.

2.2 Focal Actor and Candidate Pool

Not every event in the graph is relevant to every actor's story. We narrow the search space by fixing a protagonist and constructing the set of events that could plausibly appear in that protagonist's narrative arc.

Definition 2.5 (Focal actor)**.** The **focal actor** $a^* \in A$ is the agent whose story arc we seek to extract. An event $v \in V$ is called **focal** if $a(v) = a^*$, and **non-focal** otherwise.

Definition 2.6 (Candidate pool)**.** Given a focal actor a^* and an event graph G, the **candidate pool** $P \subseteq V$ is constructed as follows.

Step 1. Anchor selection. Choose an anchor event $e_0 \in V$ with $a(e_0) = a^*$, typically the highest-weight focal event:

$$e_0 = \operatorname*{arg\,max}_{v \in V:\, a(v) = a^*} w(v).$$

Step 2. Causal reachability. Initialize $P \leftarrow \varnothing$. Starting from e_0, perform a bidirectional BFS along the edges of G (following both incoming and outgoing causal edges) to collect all events reachable from e_0 within the causal structure. Add these events to P.

Step 3. Focal injection. Compute the focal actor's maximum-weight event:

$$e^* = \operatorname*{arg\,max}_{v \in V:\, a(v)=a^*} w(v).$$

If $e^* \notin P$, add it: $P \leftarrow P \cup \{e^*\}$.

Remark 2.7 (Structural necessity of injection). The injection in Step 3 is structurally necessary. Without it, a peripheral actor—one whose events are causally disconnected from the main action—may have an empty or nearly empty candidate pool, rendering the extraction problem trivial or vacuous. Injection guarantees that the candidate pool always contains at least one focal event, namely the highest-weight one.

However, injection creates a subtle problem: the injected event e^* may not be causally connected to the rest of the pool. This *contamination* introduces a potential semantic discontinuity in the extracted narrative. We defer the full analysis of this issue to Chapter 10.

Remark 2.8 (Focal vs. non-focal events in P). The candidate pool P typically contains both **focal events** (involving a^*) and **non-focal events** (involving other actors but causally connected to a^*'s story). Non-focal events provide context, setup, and development; focal events carry the protagonist's direct actions and the potential turning points.

2.3 The Endogenous Turning Point

The turning point—the single most dramatic moment in the protagonist's arc—is not given exogenously. It is determined by the selection itself: the turning point is the highest-weight focal event *in the selected set*. This endogenous coupling is the central structural feature of the problem.

Definition 2.9 (Endogenous turning point)**.** Given a selected subset $S \subseteq P$, the **endogenous turning point** is

$$\tau(S) = \operatorname*{arg\,max}_{v \in S:\, a(v)=a^*} w(v),$$

where ties are broken by the deterministic ordering of Definition 2.3. If S contains no focal events, $\tau(S)$ is undefined.

The critical property of the endogenous turning point is that it depends on S: changing which events are selected can change which event serves as the turning point.

Proposition 2.10 (Endogenous coupling)**.** *The turning-point function τ is not a constant of the problem instance. That is, there exist candidate pools P and subsets $S_1, S_2 \subseteq P$ such that $\tau(S_1) \neq \tau(S_2)$.*

Proof. We construct an explicit example. Let $P = \{e_1, e_2, e_3, e_4\}$ with the following attributes:

Event	Focal?	Weight	Timestamp
e_1	No	4	1
e_2	Yes	5	2
e_3	Yes	3	3
e_4	Yes	8	4

Let $S_1 = \{e_1, e_2, e_3\}$. The focal events in S_1 are e_2 (weight 5) and e_3 (weight 3), so $\tau(S_1) = e_2$.

Now let $S_2 = \{e_1, e_3, e_4\}$. The focal events in S_2 are e_3 (weight 3) and e_4 (weight 8), so $\tau(S_2) = e_4$.

Since $\tau(S_1) = e_2 \neq e_4 = \tau(S_2)$, the turning point shifted from e_2 to e_4 solely by changing the selected set. □

Remark 2.11 (Why endogeneity matters). In a standard constrained optimisation problem, the constraints are fixed functions of the decision variables. Here, the constraint *itself*—specifically, the identity of the turning point and the resulting phase labelling—is a function of the solution. This circular dependency is what makes the extraction problem fundamentally different from subset selection under monotone constraints, and it is the root cause of every pathology analysed in this book.

2.4 The Phase Grammar

A well-formed narrative arc must progress through a sequence of dramatic phases: setup, development, a turning point, and resolution. We encode this structural requirement as a deterministic finite automaton.

Definition 2.12 (Phase grammar)**.** The **phase grammar** is a DFA

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F),$$

with the following components.

(i) The **phase alphabet** is

$$\Sigma = \{ \textsc{Setup}, \textsc{Development}, \textsc{Turning\_Point}, \textsc{Resolution} \}.$$

(ii) The **state set** is

$$Q = \{ q_{\text{setup}}, q_{\text{dev}}, q_{\text{tp}}, q_{\text{res}}, q_{\text{reject}} \}.$$

(iii) The **initial state** is $q_0 = q_{\text{setup}}$.

(iv) The **accepting states** are $F = \{q_{\text{res}}\}$.

(v) The **transition function** $\delta \colon Q \times \Sigma \to Q$ is defined by the following table, where any transition not listed sends the DFA to the reject state q_{reject}, which is absorbing (all transitions from q_{reject} lead back to q_{reject}).

Current state	Input symbol	Next state	
q_{setup}	SETUP	q_{setup}	
q_{setup}	DEVELOPMENT	q_{dev}	
q_{dev}	DEVELOPMENT	q_{dev}	
q_{dev}	TURNING_POINT	q_{tp}	(only if $\geq k$ DEVELOPMENT symbols have been consumed)
q_{tp}	RESOLUTION	q_{res}	
q_{res}	RESOLUTION	q_{res}	

The parameter $k \geq 1$ is the **prefix requirement**: the minimum number of DEVELOPMENT events that must appear before the turning point.

Remark 2.13 (Monotonicity of transitions). The DFA is **monotonic**: the implicit ordering

$$q_{\text{setup}} \prec q_{\text{dev}} \prec q_{\text{tp}} \prec q_{\text{res}}$$

is never violated by a legal transition. There is no edge from q_{dev} back to q_{setup}, no edge from q_{tp} back to q_{dev}, and no edge from q_{res} to any earlier state. Phase transitions only move forward. Any input that would require regression sends the DFA to the absorbing reject state q_{reject}.

Definition 2.14 (Grammar acceptance)**.** A sequence of phase labels $\sigma_1\sigma_2\cdots\sigma_n \in \Sigma^*$ is **accepted** by $\mathcal{A}$ if and only if

$$\delta^*(q_0,\, \sigma_1\sigma_2\cdots\sigma_n) \in F,$$

where $\delta^*\colon Q \times \Sigma^* \to Q$ is the extended transition function defined recursively by $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, \sigma_1\cdots\sigma_n) = \delta^*(\delta(q, \sigma_1),\, \sigma_2\cdots\sigma_n)$.

Equivalently, a sequence is accepted if and only if it matches the pattern

$$\textsc{Setup}^*\ \textsc{Development}^{\geq k}\ \textsc{Turning\_Point}\ \textsc{Resolution}^{\geq 1}.$$

Example 2.15 (Accepted and rejected sequences)**.** Fix $k = 3$. The following sequences illustrate the grammar:

(a) S, D, D, D, TP, R: accepted (1 setup, 3 development, 1 turning point, 1 resolution).

(b) D, D, D, D, TP, R, R: accepted (0 setup, 4 development, 1 turning point, 2 resolution).

(c) S, D, D, TP, R: **rejected** — only 2 development events before the turning point, but $k = 3$ is required.

(d) S, D, D, D, TP: **rejected** — no resolution event after the turning point.

(e) S, D, TP, D, R: **rejected** — a development event appears after the turning point, violating monotonicity.

2.5 The Phase Classifier

The grammar operates on phase labels, but the raw data consists of events with timestamps and weights. A *phase classifier* bridges the gap: it assigns each selected event a phase label based on its temporal position relative to the turning point.

Definition 2.16 (Position-based phase classifier)**.** Let $S \subseteq P$ be a selected subset with endogenous turning point $\tau(S)$. Order the events of S by timestamp: $v_1, v_2, \ldots, v_n$ with $t(v_1) \leq t(v_2) \leq \cdots \leq t(v_n)$. Let $\tau(S) = v_j$ for some index j.

The **position-based phase classifier** $\varphi_\tau\colon S \to \Sigma$ assigns:

$$\varphi_\tau(v_i) = \begin{cases} \textsc{Setup} \text{ or } \textsc{Development} & \text{if } i < j, \\ \textsc{Turning\_Point} & \text{if } i = j, \\ \textsc{Resolution} & \text{if } i > j. \end{cases}$$

For events before the turning point, the distinction between SETUP and DEVELOPMENT is determined by a partition of the pre-turning-point events: the first s events (for some $s \geq 0$) are labelled SETUP, and the remaining events before the turning point are labelled DEVELOPMENT.

Definition 2.17 (Grammar-aware phase classifier)**.** The **grammar-aware phase classifier** assigns phase labels to maximise the probability of grammar acceptance. Given a selected set S and turning point $\tau(S)$, it solves:

$$\varphi^*_\tau \;=\; \arg\max_{\varphi\colon S\to\Sigma} \mathbf{1}\big[\,\delta^*(q_0,\, \varphi(v_1)\,\varphi(v_2)\,\cdots\,\varphi(v_n)) \in F\,\big],$$

subject to the constraint that $\varphi(v_j) = \textsc{Turning\_Point}$ where $v_j = \tau(S)$, and that the labelling respects the monotonic phase ordering.

In practice, this reduces to choosing the optimal number of setup events so that exactly k or more development events appear before the turning point. The grammar-aware classifier resolves Class B failures (Chapter 4) that arise when the position-based classifier produces a label sequence that the DFA rejects.

Remark 2.18 (Circular dependency). Both classifiers depend on $\tau(S)$, which in turn depends on S. The phase labels therefore depend on the selection, but the set of *valid* selections is precisely those sets whose phase labels satisfy the grammar. This circular dependency is the formal manifestation of the endogenous coupling described informally in Proposition 2.10: we cannot evaluate the constraint without knowing the solution, and we cannot construct the solution without evaluating the constraint.

2.6 The Extraction Problem

We now state the optimisation problem that the entire book studies.

Definition 2.19 (Narrative extraction problem)**.** Given an event graph $G = (V, E, t, w, a)$, a focal actor a^*, a candidate pool $P \subseteq V$, and a phase grammar $\mathcal{A}$ with prefix requirement k, the **narrative extraction problem** is:

$$S^* \;=\; \arg\max_{S\subseteq P} \sum_{v\in S} w(v) \tag{2.1}$$

subject to:

(a) **Grammar constraint.** The phase-labelled sequence $\varphi_{\tau(S)}(v_1)\,\varphi_{\tau(S)}(v_2)\,\cdots\,\varphi_{\tau(S)}(v_{|S|})$ is accepted by $\mathcal{A}$, where $v_1, \ldots, v_{|S|}$ are the events of S ordered by timestamp.

(b) **Cardinality bounds.** $n_{\min} \le |S| \le n_{\max}$, where $n_{\min}$ and $n_{\max}$ are the minimum and maximum number of beats (events) in the extracted arc.

(c) **Protagonist coverage.** The fraction of focal events in S is at least ρ:

$$\frac{|\{v \in S : a(v) = a^*\}|}{|S|} \;\ge\; \rho.$$

(d) **Timespan constraint.** The extracted arc spans at least a fraction γ of the total timeline:

$$\frac{\max_{v\in S} t(v) - \min_{v\in S} t(v)}{\max_{v\in P} t(v) - \min_{v\in P} t(v)} \;\ge\; \gamma.$$

Remark 2.20 (Endogenous constraint coupling). The grammar constraint (a) is the source of all difficulty. The constraints (b)–(d) are standard monotone set constraints: they can be checked independently of $\tau(S)$ and do not exhibit endogenous coupling. Constraint (a), by contrast, depends on S through both the selection itself (which events appear) and the turning point $\tau(S)$ (which determines the phase labelling). The extraction problem is therefore a **constrained combinatorial optimisation where the constraint function depends on the solution through** $\tau(S)$.

2.7 Solver Semantics

The extraction problem admits multiple resolution strategies, differing in how they handle the endogenous coupling. We define four solver semantics precisely. Each corresponds to a distinct algorithmic approach and to a distinct code path in the reference implementation.

2.7.1 Committed Semantics ($M = 1$)

Definition 2.21 (Committed solver)**.** The **committed solver** proceeds in two stages:

Stage 1. Pivot commitment. Compute the turning point over the *entire* candidate pool:

$$\tau^{\text{commit}} = \underset{v \in P:\, a(v)=a^*}{\arg\max}\ w(v).$$

This is the highest-weight focal event in P, determined before any selection takes place.

Stage 2. Constrained selection. Fix τ^{commit} as the turning point. Find the subset $S^* \subseteq P$ that maximises $\sum_{v \in S} w(v)$ subject to the grammar being satisfied with τ^{commit} in the TURNING_POINT role, plus all auxiliary constraints.

The pivot identity is fixed at Stage 1 and **cannot be replaced** during Stage 2.

Code path: `solve_with_budget(events, k, M=1)` in `src/compression.py`.

2.7.2 Endogenous Composition Semantics

Definition 2.22 (Endogenous composition solver)**.** Under **endogenous composition**, the pivot is not fixed in advance. Events are processed in temporal order via a left fold. At each step, the current event is composed with the running context element using the endogenous composition operator $\otimes_{\text{endo}}$ (Chapter 5). If the new event is focal and has higher weight than the current pivot, the pivot **shifts rightward** to the new event. The pivot is not locked until the full sequence has been processed.

Code path: `build_tropical_context()` in `src/tropical_semiring.py`.

2.7.3 Enumerative Semantics (Top-M)

Definition 2.23 (Enumerative solver)**.** The **enumerative solver** with parameter $M > 1$ proceeds as follows:

Step 1. Rank all focal events in P by weight. Let $c_1, c_2, \ldots, c_M$ be the top-M candidates (with $w(c_1) \geq w(c_2) \geq \cdots \geq w(c_M)$).

Step 2. For each candidate c_i, $i = 1, \ldots, M$: fix c_i as the turning point and solve the constrained extraction problem from Definition 2.19 with τ locked to c_i. Let S_i^* denote the optimal solution for candidate c_i (or $\varnothing$ if no feasible solution exists).

Step 3. Return the solution with the highest total weight:

$$S^* = \underset{i \in \{1,\ldots,M\}:\, S_i^* \neq \varnothing}{\arg\max} \sum_{v \in S_i^*} w(v).$$

Code path: `solve_with_budget(events, k, M=M)` in `src/compression.py`, with $M > 1$.

2.7.4 Fixed-Pivot Semantics

Definition 2.24 (Fixed-pivot solver)**.** The **fixed-pivot solver** is a diagnostic semantic. It constrains the solver to use the **full-sequence dominant pivot**: the turning point that would be selected if the entire candidate pool P were used without any compression or truncation. Formally:

$$\tau^{\text{fixed}} = \operatorname*{arg\,max}_{v \in P:\, a(v) = a^*} w(v).$$

The solver then finds the best $S \subseteq P$ (or $S \subseteq P'$ for a compressed pool $P' \subseteq P$) with τ^{fixed} as the mandatory turning point.

The fixed-pivot semantic answers the question: *can the original narrative meaning be preserved under this compression?* If the fixed-pivot solver produces a feasible solution, the original turning point survives. If it does not, the compression has destroyed the original meaning.

2.7.5 Relationships Between Semantics

Remark 2.25 (The validity mirage and solver semantics). The relationship between these four semantics is the source of the **validity mirage** analysed in Chapter 9. When the committed or enumerative solver with $M > 1$ produces a valid solution, the output satisfies every stated constraint. But the turning point in the output may differ from the full-sequence dominant pivot. The solution is *technically valid but semantically different* from what the full-sequence solution intended.

Concretely: the enumerative solver may find that candidate c_1 (the highest-weight focal event) cannot serve as turning point after compression—there are too few development events before it. It then falls back to candidate c_2, which *can* serve as turning point. The resulting arc is valid (it satisfies the grammar with c_2 as turning point) but tells a *different story* than the original sequence.

The fixed-pivot semantic exposes this substitution: if the fixed-pivot solver fails while the enumerative solver succeeds, a pivot substitution has occurred. This diagnostic is the principal tool for detecting the validity mirage empirically.

2.8 The Greedy Policy

The simplest concrete algorithm for the extraction problem is the greedy policy, which commits to a pivot and then fills slots in weight-descending order.

Definition 2.26 (Greedy extraction policy)**.** The **greedy policy** for the extraction problem operates as follows.

Step 1. Pivot pre-selection. Compute $e^* = \arg\max_{v \in P:\, a(v) = a^*} w(v)$. Pre-select e^* into the solution and assign it the label TURNING_POINT. Initialize $S \leftarrow \{e^*\}$.

Step 2. Candidate ranking. Sort the remaining events $P \setminus \{e^*\}$ in descending order of weight (using the deterministic tie-breaking of Definition 2.3).

Step 3. Greedy filling. For each event v in the sorted order:

(a) Determine the best-available phase label for v given the current DFA state and the temporal position of v relative to e^*.

(b) If a valid label exists (i.e., adding v with that label does not send the DFA to q_{reject}), add v to S and advance the DFA state.

(c) If no valid label exists, skip v.

Continue until $|S| = n_{\max}$ or all candidates have been considered.

Step 4. No backtracking. Once an event is selected and labelled, the decision is **irrevocable**. The greedy policy never reconsiders a previous choice.

Remark 2.27 (Greedy as committed $M = 1$). The greedy policy is a **committed ($M = 1$) strategy** with the additional property that it makes locally optimal weight choices at each step. The pivot is fixed at Step 1 (committed semantics), and the remaining selections are made greedily. The absence of backtracking means the greedy policy can fail to find a feasible solution even when one exists: it may consume all available development slots on high-weight events that are temporally positioned after the turning point, leaving too few development events before the turning point to satisfy the prefix requirement k.

Definition 2.28 (Bounded-prefix phase grammar)**.** The **bounded-prefix phase grammar** is the phase grammar of Definition 2.12 augmented with an upper bound $k_{\max} \geq k_{\min}$ on the DEVELOPMENT phase: a run is accepted only if the number of DEVELOPMENT symbols consumed before TURNING_POINT lies in the closed interval $[k_{\min}, k_{\max}]$. The lower bound $k_{\min}$ is the standard prefix requirement k of Definition 2.12; the upper bound $k_{\max}$ prevents excessively long setup portions of the narrative arc.

2.9 Non-Matroid Structure

The greedy policy has no performance guarantee for the extraction problem. This is not an artefact of our particular greedy strategy; it is a structural property of the problem itself. The feasible family violates the axioms that would be needed to guarantee greedy optimality.

Definition 2.29 (Feasible family)**.** The **feasible family** for the extraction problem is

$$\mathcal{F} \;=\; \big\{\, S \subseteq P \;:\; \text{the phase-labelled sequence of } S \text{ is accepted by } \mathcal{A} \,\big\}.$$

That is, $\mathcal{F}$ is the collection of all subsets of P whose induced phase labelling (using the endogenous turning point $\tau(S)$ and the grammar-aware classifier) satisfies the phase grammar.

We recall the two axioms that define a matroid on ground set P.

- **Hereditary axiom.** If $S \in \mathcal{F}$ and $S' \subseteq S$, then $S' \in \mathcal{F}$.
- **Exchange axiom.** If $S_1, S_2 \in \mathcal{F}$ with $|S_1| < |S_2|$, then there exists $v \in S_2 \setminus S_1$ such that $S_1 \cup \{v\} \in \mathcal{F}$.

Proposition 2.30 (Non-matroid structure)**.** *The feasible family $\mathcal{F}$ violates both the hereditary axiom and the exchange axiom. Consequently, $\mathcal{F}$ is not a matroid, not a greedoid [2, 14], and not a polymatroid.*

Proof. We prove each violation separately with explicit constructions.

Part 1: Violation of the hereditary axiom.

Fix $k = 3$. Define the following six events, where the focal actor is a^*:

Event	Actor	Weight	Timestamp	Intended role
e_s	other	1	1	Setup
e_{d1}	other	2	2	Development
e_{d2}	other	2	3	Development
e_{d3}	other	2	4	Development
e_{tp}	a^*	9	5	Turning point
e_r	other	3	6	Resolution

Consider the full set

$$S = \{e_s, e_{d1}, e_{d2}, e_{d3}, e_{tp}, e_r\}.$$

The event e_{tp} is the sole focal event in S, so $\tau(S) = e_{tp}$. The phase-labelled sequence (in temporal order) is

$$\underbrace{\text{S}}_{e_s}\ \underbrace{\text{D}}_{e_{d1}}\ \underbrace{\text{D}}_{e_{d2}}\ \underbrace{\text{D}}_{e_{d3}}\ \underbrace{\text{TP}}_{e_{tp}}\ \underbrace{\text{R}}_{e_r}.$$

This has 1 setup, 3 development events (meeting $k = 3$), 1 turning point, and 1 resolution. The DFA accepts. Therefore $S \in \mathcal{F}$.

Now remove e_{d1} to form $S' = S \setminus \{e_{d1}\}$. The turning point is still $\tau(S') = e_{tp}$ (still the unique focal event). The phase-labelled sequence becomes

$$\underbrace{\text{S}}_{e_s}\ \underbrace{\text{D}}_{e_{d2}}\ \underbrace{\text{D}}_{e_{d3}}\ \underbrace{\text{TP}}_{e_{tp}}\ \underbrace{\text{R}}_{e_r}.$$

Now there are only 2 development events before the turning point. The grammar-aware classifier cannot improve on this: regardless of whether e_s is labelled SETUP or DEVELOPMENT, the maximum number of development events before e_{tp} is 3 (if e_s is relabelled as DEVELOPMENT)—but wait: if we relabel e_s as DEVELOPMENT, we obtain D, D, D, TP, R, which has exactly 3 development events and satisfies $k = 3$.

To close this escape route, we remove *two* development events. Let $S'' = S \setminus \{e_{d1}, e_{d2}\}$:

$$S'' = \{e_s, e_{d3}, e_{tp}, e_r\}.$$

$\tau(S'') = e_{tp}$. Before e_{tp}: events e_s and e_{d3}. Even relabelling both as DEVELOPMENT, we get only 2 development events. Since $k = 3$, the DFA rejects. $S'' \notin \mathcal{F}$.

We have $S \in \mathcal{F}$ and $S'' \subset S$ with $S'' \notin \mathcal{F}$. The hereditary axiom is violated.

Part 2: Violation of the exchange axiom.

We use the bounded-prefix phase grammar (Definition 2.28) with $k_{\min} = 3$ and $k_{\max} = 4$: at least 3 and at most 4 DEVELOPMENT symbols must appear before the turning point.

Define the following events:

Event	Actor	Weight	Timestamp
c_1	other	1	1
c_2	other	1	2
c_3	other	1	3
q	a^*	9	3.5
c_4	other	1	4
c_5	other	1	5
p	a^*	7	6
r	other	1	7

Feasible set S_1. Let $S_1 = \{c_1, c_3, c_4, c_5, p, r\}$, with $|S_1| = 6$. The only focal event is p (weight 7), so $\tau(S_1) = p$ (timestamp 6). Events before p in temporal order: $c_1(1)$, $c_3(3)$, $c_4(4)$, $c_5(5)$—all non-focal, all labelled DEVELOPMENT. That gives 4 development events: $k_{\min} = 3 \leq 4 \leq k_{\max} = 4$. After p: r labelled RESOLUTION. Phase sequence: D, D, D, D, TP, R. Accepted. $S_1 \in \mathcal{F}$.

Feasible set S_2. Let $S_2 = \{c_1, c_2, c_3, q, c_4, c_5, p, r\}$, with $|S_2| = 8$. Focal events: q (weight 9) and p (weight 7). $\tau(S_2) = q$ (timestamp 3.5). Events before q: $c_1(1)$, $c_2(2)$, $c_3(3)$—3 development events. $k_{\min} = 3 \leq 3 \leq k_{\max} = 4$. After q: c_4, c_5, p, r all labelled RESOLUTION. Phase sequence: D, D, D, TP, R, R, R, R. Accepted. $S_2 \in \mathcal{F}$.

Exchange check. We have $|S_1| = 6 < 8 = |S_2|$ and $S_2 \setminus S_1 = \{c_2, q\}$. The exchange axiom requires the existence of some $v \in \{c_2, q\}$ with $S_1 \cup \{v\} \in \mathcal{F}$. We show that neither works.

(i) **Adding** q (focal, weight 9, timestamp 3.5) to S_1. The augmented set is $S_1 \cup \{q\} = \{c_1, c_3, q, c_4, c_5, p, r\}$. Now $\tau(S_1 \cup \{q\}) = q$ because $w(q) = 9 > 7 = w(p)$—the pivot shifts from p to q. Events before q (timestamp 3.5): $c_1(1)$ and $c_3(3)$—only 2 non-focal events. Even labelling both as DEVELOPMENT, we have $2 < k_{\min} = 3$. The prefix requirement is **not met**. The DFA rejects: $S_1 \cup \{q\} \notin \mathcal{F}$.

(ii) **Adding** c_2 (non-focal, timestamp 2) to S_1. The pivot remains $\tau(S_1 \cup \{c_2\}) = p$ (no new focal events). Events before p: $c_1(1)$, $c_2(2)$, $c_3(3)$, $c_4(4)$, $c_5(5)$ —5 development events. But $k_{\max} = 4$, and $5 > 4$. The grammar **rejects**: too many development events before the turning point. $S_1 \cup \{c_2\} \notin \mathcal{F}$.

Neither element of $S_2 \setminus S_1$ can be added to S_1 to yield a feasible set. The exchange axiom is violated. □

Remark 2.31 (Bounded-prefix scope). The exchange-axiom failure above is specific to the bounded-prefix grammar $d_{\text{pre}} \geq k$. Different grammars (e.g., a flat "select any m events" objective) may or may not exhibit a matroid structure; the non-matroid conclusion does not extend beyond the class of grammars with positional requirements relative to the pivot.

Remark 2.32 (Consequences of non-matroid structure). Proposition 2.30 has three immediate consequences:

(1) **No greedy guarantee.** The classical result that the greedy algorithm is optimal for matroids (and, more generally, for greedoids) does not apply. The greedy policy of Definition 2.26 may produce arbitrarily suboptimal solutions.

(2) **No exchange property.** The exchange axiom failure means there is no guarantee that a small feasible set can be augmented element-by-element to a larger feasible set. Local augmentation strategies—which underlie many combinatorial optimisation algorithms—cannot be relied upon.

(3) **No polymatroid structure.** Since the hereditary axiom fails, the feasible family is not even a simplicial complex, let alone a matroid or polymatroid. The extraction problem falls outside the scope of submodular optimisation theory.

These failures motivate the algorithmic hierarchy developed in Chapter 4: since no generic optimisation framework applies, we must develop problem-specific methods tailored to the endogenous coupling structure.

Chapter 3

Absorbing States in Greedy Search

The greedy policy introduced in Chapter 2 selects a turning point by taking the arg max over focal-actor weights and then fills the remaining narrative phases around it. This chapter shows that a simple counting argument—comparing the number of development-eligible events against the grammar's prefix requirement—yields an exact impossibility boundary. Below this boundary, the greedy policy produces *zero* valid sequences; the product of the event graph with the phase grammar enters an absorbing state from which no continuation can reach acceptance.

We proceed in stages. Section 3.1 formalises the development-eligible set and its count j_{dev}. Section 3.2 specifies the phase grammar as a deterministic finite automaton and proves its monotonicity lemma. Section 3.3 constructs the product automaton that couples the greedy walk with the grammar. Section 3.4 states and proves the central result, Theorem 3.8, in five self-contained steps. Section 3.5 collects important remarks on the theorem's scope. Section 3.6 presents the empirical evidence, and Section 3.7 offers exercises.

3.1 Development-Eligible Events

Throughout, let $G = (V, E, a, t, w)$ be an event graph on vertex set V, where $a(v)$ is the actor of event v, $t(v)$ its timestamp, and $w(v)$ its narrative weight. Fix a focal actor a^* and write

$$e^* \;=\; \arg\max_{v\,:\,a(v)=a^*} w(v)$$

for the *forced turning point*—the event selected by the greedy policy as the structural pivot. Let $k \geq 1$ denote the grammar's *prefix requirement*: the minimum number of DEVELOPMENT symbols that must be consumed before TURNING_POINT.

Definition 3.1 (Development-eligible set)**.** Given the event graph G, turning point e^*, and prefix requirement k, the *candidate pool* is $P = V \setminus \{e^*\}$. The *development-eligible set* is

$$D \;=\; \left\{\, v \in P \;\middle|\; t(v) < t(e^*) \text{ and } v \text{ can receive the DEVELOPMENT label under the phase grammar} \,\right\}.$$

The *development-eligible count* is $j_{\text{dev}} = |D|$.

The two conditions in the definition deserve emphasis. Condition (i), $t(v) < t(e^*)$, restricts attention to events that precede the turning point in the timeline; events after the turning point cannot contribute DEVELOPMENT symbols under the monotonic grammar (Lemma 3.6). Condition (ii) encodes the classifier–policy interaction: whether a given event *can* receive the DEVELOPMENT label depends on the classifier in use. (The exclusion $v \neq e^*$ is already captured by the requirement $v \in P = V \setminus \{e^*\}$.)

Remark 3.2 (Classifier dependence of j_{dev}). The count j_{dev} depends on the classifier–policy interaction. Under a *position-based classifier*, any event before the turning point in the timeline is eligible for DEVELOPMENT, so $j_{\text{dev}} = |\{v \in P : t(v) < t(e^*)\}|$. Under a *grammar-aware classifier*, only events that the DFA can label as DEVELOPMENT—for instance, events belonging to non-focal actors—are eligible, and j_{dev} may be strictly smaller. This distinction is the source of Class B failures analysed in Chapter 4.

Example 3.3 (Position-based vs. grammar-aware counting)**.** Consider a sequence of $n = 10$ events in which the turning point e^* occupies position 7 (in temporal order). Among the six events preceding e^*, two belong to the focal actor a^* and four belong to other actors.

(i) **Position-based classifier.** Every event before the turning point is eligible: $j_{\text{dev}} = 6$.

(ii) **Grammar-aware classifier** (requiring DEVELOPMENT events to be non-focal). Only the four non-focal events qualify: $j_{\text{dev}} = 4$.

If the prefix requirement is $k = 5$, the position-based classifier gives $j_{\text{dev}} = 6 \geq k$ (no impossibility predicted), while the grammar-aware classifier gives $j_{\text{dev}} = 4 < k$ (Theorem 3.8 applies: greedy validity is exactly zero).

3.2 The Phase Grammar DFA

We model the narrative phase grammar as a deterministic finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ defined as follows.

Definition 3.4 (Phase grammar DFA)**.**

- **States.** $Q = \{S_{\text{SETUP}}, S_{\text{DEV}}, S_{\text{TP}}, S_{\text{RES}}, S_{\text{REJECT}}\}$.
- **Alphabet.** $\Sigma = \{\text{SETUP}, \text{DEVELOPMENT}, \text{TURNING\_POINT}, \text{RESOLUTION}\}$.
- **Initial state.** $q_0 = S_{\text{SETUP}}$.
- **Accepting states.** $F = \{S_{\text{RES}}\}$.
- **Transition function δ.** The transitions are *monotonic*: phase labels must appear in non-decreasing order SETUP $\prec$ DEVELOPMENT $\prec$ TURNING_POINT $\prec$ RESOLUTION. Formally, the non-reject transitions are:

Current state	Symbol	Next state
S_{SETUP}	SETUP	S_{SETUP}
S_{SETUP}	DEVELOPMENT	S_{DEV}
S_{DEV}	DEVELOPMENT	S_{DEV}
S_{DEV}	TURNING_POINT	S_{TP}
S_{TP}	RESOLUTION	S_{RES}
S_{RES}	RESOLUTION	S_{RES}

Every (q, σ) pair not listed above transitions to S_{REJECT}. The state S_{REJECT} is *absorbing*: $\delta(S_{\text{REJECT}}, \sigma) = S_{\text{REJECT}}$ for every $\sigma \in \Sigma$.

- **Prefix requirement.** The automaton must consume at least k copies of the symbol DEVELOPMENT before consuming TURNING_POINT for the run to be valid. Equivalently, one may augment the state space with a counter $c \in \{0, 1, \ldots, k\}$ and allow the transition $S_{\text{DEV}} \xrightarrow{\text{TURNING\_POINT}} S_{\text{TP}}$ only when $c \geq k$. For readability, we keep the counter implicit and treat the prefix requirement as a side condition on acceptance.

Remark 3.5 (Consistency with Definition 2.12). This table is identical to the one in Definition 2.12 of Chapter 2. In particular, the transition $S_{\text{SETUP}} \xrightarrow{\text{TURNING\_POINT}} S_{\text{TP}}$ is *not* listed: a sequence that reaches the turning point without any preceding development symbol immediately violates the prefix requirement k and is rejected via the counter mechanism. The k-counter enforces the minimum-development condition; the unlisted transition is the structural enforcement.

The key structural property of this DFA is that phases advance monotonically and can never retreat. The following lemma isolates the consequence that matters most for the impossibility theorem.

Lemma 3.6 (DFA Monotonicity)**.** *Any state reached after consuming* TURNING_POINT *is absorbing with respect to* DEVELOPMENT *transitions. Formally, if*

$$\delta^*(q_0,\ \sigma_1 \cdots \sigma_m)\ \in\ \{S_{\text{TP}},\ S_{\text{RES}}\},$$

then for every continuation $\sigma_{m+1} \cdots \sigma_n$ *with* $\sigma_j =$ DEVELOPMENT *for some* $j \in \{m+1, \ldots, n\}$*, we have*

$$\delta^*(q_0,\ \sigma_1 \cdots \sigma_n)\ =\ S_{\text{REJECT}}.$$

Proof. We argue by inspection of the transition table in Definition 3.4.

Case 1: the DFA is in S_{TP}. The only symbol that leads to a non-reject state is RESOLUTION, which transitions to S_{RES}. Consuming DEVELOPMENT in S_{TP} yields S_{REJECT}.

Case 2: the DFA is in S_{RES}. The only symbol that keeps the DFA in a non-reject state is RESOLUTION (self-loop on S_{RES}). Consuming DEVELOPMENT in S_{RES} yields S_{REJECT}.

Case 3: the DFA is in S_{REJECT}. By definition, S_{REJECT} is absorbing: $\delta(S_{\text{REJECT}}, \sigma) = S_{\text{REJECT}}$ for all $\sigma \in \Sigma$.

In every case, once TURNING_POINT has been consumed, the DFA is in S_{TP}, S_{RES}, or S_{REJECT}. From any of these states, consuming DEVELOPMENT leads (immediately or eventually) to S_{REJECT}, from which acceptance is impossible. Therefore the DFA never returns to S_{DEV} after processing TURNING_POINT. □

3.3 The Product Automaton

The impossibility proof couples the greedy policy's event-selection order with the grammar DFA. The vehicle for this coupling is a *product automaton* $\mathcal{P}$.

Definition 3.7 (Product automaton)**.** Let $G = (V, E, a, t, w)$ be an event graph with $|V| = n$, and let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be the phase grammar DFA (Definition 3.4). Linearise the event graph by the greedy policy's selection order: events are processed in decreasing weight order, $w(v_1) \geq w(v_2) \geq \cdots \geq w(v_n)$, with ties broken by a fixed deterministic rule (e.g., earliest timestamp first).

The *product automaton* is $\mathcal{P} = G \times \mathcal{A}$, with:

- **State space.** Pairs $(i,\, q)$ where $i \in \{0, 1, \ldots, n\}$ indexes the current position in the greedy selection order and $q \in Q$ is the DFA state.
- **Initial state.** $(0,\, q_0) = (0,\, S_{\text{SETUP}})$.
- **Transitions.** At position i, the greedy policy selects event v_{i+1} and assigns it the "best" phase label $\sigma_{i+1} \in \Sigma$ that keeps the DFA on a path toward acceptance. The product automaton transitions from (i, q) to $(i{+}1,\ \delta(q, \sigma_{i+1}))$.
- **Acceptance.** The product automaton accepts if and only if it reaches a state (n, q_f) with $q_f \in F = \{S_{\text{RES}}\}$ and the prefix requirement (at least k DEVELOPMENT symbols before TURNING_POINT) is satisfied.

Note that the greedy walk selects events in weight order, but phase labels are assigned relative to the turning point's timestamp, which is fixed at selection time. An event classified as DEVELOPMENT during greedy selection always has timestamp earlier than the turning point's timestamp, by the event-ordering convention of Definition 2.3.

The greedy policy imposes one hard constraint on the walk through $\mathcal{P}$:

> **Turning-point pre-selection.** The event $e^* = \arg\max_{v:a(v)=a^*} w(v)$ is pre-selected as the turning point. When the walk reaches the position i^* at which e^* appears in the weight-sorted order, the label must be $\sigma_{i^*} = \text{TURNING\_POINT}$.

Between positions 0 and $i^* - 1$, the greedy policy assigns each event the best available label—typically SETUP or DEVELOPMENT—that advances the DFA toward the prefix requirement. At position i^*, the DFA consumes TURNING_POINT and transitions to S_{TP}. From position $i^* + 1$ onward, the only productive label is RESOLUTION, by Lemma 3.6.

The product automaton makes the proof's state-space argument explicit: every run of the greedy policy corresponds to a unique path through $\mathcal{P}$, and the question of whether the greedy policy can produce a valid sequence reduces to whether there exists an accepting path in $\mathcal{P}$.

3.4 The Prefix-Constraint Impossibility Theorem

We are now in a position to state and prove the chapter's main result.

Theorem 3.8 (Prefix-Constraint Impossibility)**.** *Let G be an event graph, e^* the forced turning point, k the prefix requirement, and j_{dev} the development-eligible count (Definition 3.1). If*

$$j_{\text{dev}} \;<\; k,$$

then the greedy policy produces zero valid sequences.

Proof. The proof proceeds in five steps.

Step 1 (Product automaton). Construct the product automaton $\mathcal{P} = G \times \mathcal{A}$ as in Definition 3.7. Every run of the greedy policy traces a unique path through the state space $\{0, \ldots, n\} \times Q$. The greedy policy produces a valid sequence if and only if this path ends in an accepting state (n, q_f) with $q_f \in F$ and the prefix requirement satisfied.

Step 2 (Turning-point pre-selection). The greedy policy pre-selects $e^* = \arg\max_{v:a(v)=a^*} w(v)$ as the turning point. Let i^* be the position of e^* in the weight-sorted selection order. At step i^*, the policy assigns the label $\sigma_{i^*} = \text{TURNING\_POINT}$, and the DFA transitions to S_{TP}:

$$\delta(q_{i^*-1},\ \text{TURNING\_POINT}) \;=\; S_{\text{TP}},$$

where q_{i^*-1} is the DFA state upon entering step i^* (necessarily S_{SETUP} or S_{DEV} if the run has not already reached S_{REJECT}).

Step 3 (Post-TP absorption). By the DFA Monotonicity Lemma (Lemma 3.6), once the DFA enters S_{TP}, it can never return to S_{DEV}. Any subsequent DEVELOPMENT symbol sends the DFA to S_{REJECT}, from which acceptance is impossible. Therefore, *all* DEVELOPMENT symbols must be consumed *before* step i^*.

Step 4 (Counting development slots). The events available to receive DEVELOPMENT labels before step i^* are exactly the members of the development-eligible set D from Definition 3.1. Each such event can contribute at most one DEVELOPMENT symbol to the run. Therefore, the maximum number of DEVELOPMENT symbols consumed before TURNING_POINT is

$$\#\{\text{DEVELOPMENT before TURNING\_POINT}\} \;\leq\; |D| \;=\; j_{\text{dev}}.$$

Step 5 (Prefix violation). The grammar requires at least k DEVELOPMENT symbols before TURNING_POINT for the run to satisfy the prefix requirement. By Step 4, at most j_{dev} such symbols can appear. If $j_{\text{dev}} < k$, then

$$\#\{\text{DEVELOPMENT before TURNING\_POINT}\} \;\leq\; j_{\text{dev}} \;<\; k.$$

The prefix requirement is therefore violated on *every* path through $\mathcal{P}$. No path can reach an accepting state with a satisfied prefix requirement. Equivalently, the product automaton enters an absorbing region—no matter how the remaining events are labelled, acceptance is unreachable.

Since every run of the greedy policy corresponds to a path through $\mathcal{P}$, and no such path is accepting when $j_{\text{dev}} < k$, the greedy policy produces zero valid sequences. □

3.5 Important Remarks

Remark 3.9 (Converse is false). The condition $j_{\text{dev}} \geq k$ does *not* guarantee that the greedy policy succeeds. Having enough development-eligible events is *necessary* for validity but far from *sufficient*. The greedy policy may still fail for independent reasons: it may mis-order events so that the DFA enters S_{REJECT} through a label conflict (Class C failure), or it may select a turning point that is semantically incoherent even though the grammar is satisfied (Class D failure). Both of these failure modes are analysed in Chapter 4.

Theorem 3.8 therefore provides a *sufficient condition for failure*, not a necessary one. The failure boundary $j_{\text{dev}} = k$ is sharp in one direction only: below it, validity is exactly zero; above it, validity is possible but not guaranteed.

Remark 3.10 (Focal-only counting produces false positives). A common error is to count only focal-actor events (those with $a(v) = a^*$) as development-eligible, ignoring events from other actors. This under-counts the development-eligible set: if an event v with $a(v) \neq a^*$ can receive the DEVELOPMENT label, it belongs in D.

Focal-only counting yields ${j_{\text{dev}}}^{\text{focal}} \leq j_{\text{dev}}$, which may trigger a false positive prediction of impossibility: the focal-only count may satisfy ${j_{\text{dev}}}^{\text{focal}} < k$ even when $j_{\text{dev}} \geq k$. The correct count uses *all* events eligible for DEVELOPMENT, regardless of actor. This point is validated by the test assertions in the repository's test suite, which confirm that multi-actor events contribute to the development-eligible pool.

Remark 3.11 (Bridge to d_{pre}). In the *unit-mass regime*—where each event contributes exactly one unit of development content—the development-eligible count coincides with the pre-pivot development mass: $j_{\text{dev}} = d_{\text{pre}}$. The absorbing-state condition $j_{\text{dev}} < k$ then becomes $d_{\text{pre}} < k$, which is precisely the *absorbing predicate* $\perp$ from the context algebra developed in Chapter 7.

This bridge connects the combinatorial argument of the present chapter with the algebraic framework: the absorbing ideal of Chapter 7 is the algebraic closure of the impossibility region identified here. When event masses are non-uniform, j_{dev} and d_{pre} may diverge, and the algebraic formulation (via d_{pre}) provides the more refined characterisation.

3.6 Empirical Validation

The theoretical boundary $j_{\text{dev}} = k$ admits clean empirical verification. We summarise the main findings [4]; full experimental details appear in the repository.

The boundary heatmap. Figure 3.1 displays the greedy policy's validity rate as a function of the prefix requirement k (horizontal axis) and the development-eligible count j_{dev} (vertical axis), aggregated over 11,400 event-graph instances. The k–j_{dev} plane divides into three clearly delineated regions:

(i) **Impossibility zone** ($j_{\text{dev}} < k$, below the diagonal). Greedy validity is exactly 0.000 across all 11,400 instances in this region, confirming Theorem 3.8 without exception.

(ii) **Stochastic zone** ($k \leq j_{\text{dev}} \lesssim 5.31k + 12.23$). Validity rises stochastically from 0 toward 1.0. In this band, having enough development-eligible events is necessary but not sufficient; other failure modes (Classes C and D) depress the success rate.

(iii) **High-validity zone** ($j_{\text{dev}} \gtrsim 5.31k + 12.23$). The 95% success envelope lies approximately along the line $j_{\text{dev}} \approx 5.31k + 12.23$. Above this envelope, the greedy policy almost always succeeds.

Exogenous turning-point control experiment. A natural question is whether the impossibility barrier is an artifact of the grammar alone or whether it depends on the greedy policy's specific method of turning-point selection. To test this, we repeat the experiment with an *exogenous* turning point: instead of selecting $e^* = \arg\max_{v:a(v)=a^*} w(v)$, we fix the turning point at the median-timestamp focal event (rather than the max-weight focal event).

Under exogenous assignment, the diagonal barrier persists *structurally*—the proof of Theorem 3.8 does not depend on *how* e^* is chosen, only on the relationship between j_{dev} and k once e^* is fixed. However, the *probability* that a randomly generated instance falls below the diagonal changes dramatically: $P(j_{\text{dev}} < k)$ drops from 13.6% under the greedy (argmax) policy to 0.0% under exogenous assignment.

The absorbing state is therefore activated by the *interaction* of weight-based turning-point selection with temporal front-loading of high-weight events—not by the grammar alone. When the turning point is the highest-weight focal event, it tends to sit early in the weight-sorted order, leaving few events before it in the timeline. The grammar's prefix requirement then becomes unsatisfiable. Exogenous assignment breaks this coupling, eliminating the pathway into the impossibility zone.

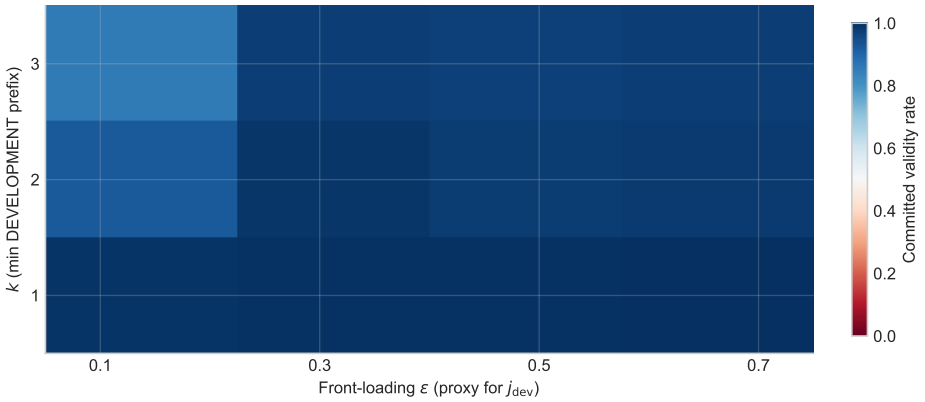


Figure 3.1: Greedy validity rate in the k–j_{dev} plane (11,400 instances). The diagonal $j_{\text{dev}} = k$ (dashed white line) is the impossibility boundary of Theorem 3.8: every cell below the diagonal has validity exactly 0.000. The solid white curve marks the 95% success envelope, approximately $j_{\text{dev}} \approx 5.31k + 12.23$. Three regions emerge: the *impossibility zone* (below diagonal), the *stochastic zone* (between diagonal and envelope), and the *high-validity zone* (above envelope).

3.7 Exercises

Exercise 3.1. Construct a minimal event graph with $n = 8$ events where Theorem 3.8 predicts failure (i.e., $j_{\text{dev}} < k$ for $k = 3$) and verify by exhaustive enumeration that no valid sequence exists under the greedy policy.

Hint. Choose weights so that e^* (the maximum-weight focal event) is at position 2 in the timeline, leaving at most two events before it. Then $j_{\text{dev}} \leq 2 < 3 = k$. To verify, enumerate all possible label assignments to the eight events and confirm that every assignment either violates the phase grammar's monotonicity, fails the prefix requirement, or both.

Exercise 3.2. Show that relaxing the prefix requirement from $k = 3$ to $k = 1$ changes the location of the impossibility boundary but not the structure of the proof.

(a) State the new impossibility condition.

(b) Identify which step of the proof changes and which steps remain identical.

(c) Give an example of an event graph that is impossible under $k = 3$ but valid under $k = 1$.

Solution sketch. The new impossibility condition is $j_{\text{dev}} < 1$, i.e., $j_{\text{dev}} = 0$: there are no development-eligible events before the turning point. Steps 1–4 of the proof are identical; only Step 5 changes, replacing the bound $j_{\text{dev}} < 3$ with $j_{\text{dev}} < 1$. Any event graph with $1 \leq j_{\text{dev}} < 3$ is impossible under $k = 3$ but potentially valid under $k = 1$.

Exercise 3.3. Prove that the product automaton $\mathcal{P}$ from Definition 3.7 has at most $|V| \times |Q|$ states, giving an $O(n)$ bound on the state-space size (since $|Q| = 5$ is constant).

Solution. The state space of $\mathcal{P}$ is $\{0, 1, \ldots, n\} \times Q$, which has $(n+1) \times |Q|$ elements. Since $|Q| = 5$ is a fixed constant independent of the event graph,

$$|\text{States of } \mathcal{P}| \;=\; (n+1) \cdot 5 \;=\; 5n + 5 \;\in\; O(n).$$

Each step of the walk through $\mathcal{P}$ processes one event and performs a constant-time DFA transition, so the entire walk takes $O(n)$ time. The proof of Theorem 3.8 thus operates on a linear-size state

space, and the counting argument in Steps 4–5 requires no search—only a comparison between j_{dev} and k, both computable in $O(n)$ time by a single pass over the event graph. □

Exercise 3.4. Consider a *modified* phase grammar in which DEVELOPMENT events may appear after TURNING_POINT. Specifically, add the transitions

$$\delta(S_{\text{TP}}, \text{DEVELOPMENT}) = S_{\text{TP}} \quad \text{and} \quad \delta(S_{\text{RES}}, \text{DEVELOPMENT}) = S_{\text{RES}}$$

to the DFA of Definition 3.4.

(a) Does Theorem 3.8 still hold under this modified grammar?

(b) Identify the exact step in the proof that breaks and explain why.

(c) Give a concrete event graph where $j_{\text{dev}} < k$ under the original grammar's definition of D, but the modified grammar admits a valid sequence.

Solution sketch.

(a) No. Theorem 3.8 does *not* hold under the modified grammar.

(b) Step 3 of the proof relies on the DFA Monotonicity Lemma (Lemma 3.6), which states that no DEVELOPMENT symbol can be consumed after TURNING_POINT. The modified grammar explicitly violates this property: DEVELOPMENT is accepted in both S_{TP} and S_{RES}. With monotonicity broken, DEVELOPMENT symbols can be consumed *after* the turning point, so the count of development slots is no longer bounded by the pre-TP count j_{dev}. Step 4's upper bound becomes invalid, and Step 5's conclusion no longer follows.

(c) Consider an event graph with $n = 6$, $k = 3$, e^* at position 2 in the timeline (so $j_{\text{dev}} = 1 < 3$). Under the original grammar, the greedy policy cannot produce a valid sequence. Under the modified grammar, one can place one DEVELOPMENT event before e^* and two DEVELOPMENT events after it, satisfying the prefix requirement of $k = 3$ total DEVELOPMENT symbols (now allowed both before and after the turning point). □

Chapter 4

Failure Taxonomy and Constructive Hierarchy

The absorbing-state theorem of Chapter 3 gives a sharp sufficient condition for greedy failure: when $j_{\text{dev}} < k$, no valid sequence exists. But the converse is false (Remark 3.9). When $j_{\text{dev}} \geq k$, the greedy policy may still fail—and it does so in structurally distinct ways. This chapter organises those failure modes into a four-class taxonomy, shows that their fixes interact antagonistically, and constructs a hierarchy of increasingly capable algorithms that attack the failure classes in sequence.

We proceed as follows. Section 4.1 presents the taxonomy (Classes A–D) with a summary table and worked examples. Section 4.2 documents the surprising result that fixing one failure class can worsen another, establishing a partial order among solvers. Section 4.3 lays out the full algorithm hierarchy from myopic greedy to oracle. Section 4.4 details the TP-conditioned solver (Algorithm 3) that occupies the practically important level of the hierarchy. Section 4.5 derives the failure decomposition formula that connects the taxonomy to the absorbing-state theory. Section 4.6 analyses approximation quality, confirming that the landscape is feasibility-dominated. Section 4.7 offers exercises.

4.1 Failure Classes

The empirical analysis of 138 greedy failures on bursty event-graph instances reveals four structurally distinct failure mechanisms. Each class is defined by the stage of the pipeline at which the failure manifests, the structural invariant that is violated, and the minimal algorithmic intervention that repairs it.

Table 4.1 summarises the four classes. We now examine each in detail.

4.1.1 Class A: Absorbing State Failures

Class A failures are the domain of Theorem 3.8. The mechanism is simple counting: the greedy policy selects a turning point e^* that sits so early in the timeline—or, equivalently, that has so few non-focal events preceding it—that the development-eligible count j_{dev} falls below the grammar's prefix requirement k.

Example 4.1 (Class A failure)**.** Let $k = 3$ and suppose the turning point e^* occupies position 0.1 on the normalised timeline (i.e., the first decile). If only two non-focal events precede e^*, then

Table 4.1: Failure taxonomy for greedy narrative-arc construction. Prevalence figures refer to the bursty event-graph domain (138 greedy failures total). Recovery rates are measured against the class-specific fix applied in isolation.

Class	Name	Mechanism	Prevalence	Fix	Recovery
A	Absorbing State	$j_{\text{dev}} < k$: turning point selected before enough development-eligible events exist. Structural impossibility—no rearrangement can produce a valid sequence.	13.6%	TP reassignment	—
B	Pipeline Coupling	Phase classifier misassigns labels. Position-based classification ignores the DFA state, mislabelling events that could serve as DEVELOPMENT.	45.7%	Grammar-aware classifier	100%
C	Commitment Timing	Greedy policy commits to bridge or span-critical events too early/late. Wrong commitment window skips infrastructure events needed for temporal coverage.	13.0%	TP-conditioned solver	85–81%
D	Assembly Compression	Weight-maximising selection compresses phase structure. High-weight events crowd into RESOLUTION, starving DEVELOPMENT.	41.3%	Span-VAG with span filtering	96.5%

$j_{\text{dev}} = 2 < 3 = k$. By Theorem 3.8, no valid sequence exists under this pivot assignment. The product automaton enters an absorbing region from which no continuation can reach acceptance.

The constructive fix is *TP reassignment*: select a different focal event as the pivot—one that sits later in the timeline and affords $j_{\text{dev}} \geq k$. This is the strategy employed by the TP-conditioned solver of Section 4.4.

Class A failures account for 13.6% of bursty instances and represent the hard boundary predicted by theory. No algorithmic improvement to the *inner* solver can repair them; only a change of pivot (an *outer-loop* intervention) suffices.

4.1.2 Class B: Pipeline Coupling Failures

Class B failures arise not from the grammar itself but from the *classifier* that maps events to phase labels. A position-based classifier assigns labels based on an event's temporal position relative to fixed thresholds (e.g., the first third of the timeline is SETUP, the middle third is DEVELOPMENT). This classifier is blind to the DFA's current state: it may label an event as SETUP even though the DFA has already transitioned to S_{DEV} and would accept a DEVELOPMENT symbol.

Example 4.2 (Class B failure)**.** Consider an event at normalised position 0.05 in the timeline. A position-based classifier labels it SETUP because it falls in the earliest segment. However, the DFA has already consumed a DEVELOPMENT symbol (from a preceding event) and is in state S_{DEV}. A grammar-aware classifier would recognise this and correctly label the event as DEVELOPMENT, contributing to the prefix requirement. The position-based classifier wastes a development opportunity, potentially pushing j_{dev} below k.

Class B failures are the most prevalent single class, accounting for 63 of 138 failures (45.7%). They are also the most cleanly repairable: a grammar-aware classifier that tracks the DFA state recovers 63/63 (100%) of Class B failures. This perfect recovery rate reflects the fact that the underlying grammar structure is sound; only the label assignment was wrong.

4.1.3 Class C: Commitment Timing Failures

Class C failures occur when the greedy policy's commitment order—the sequence in which it irrevocably selects events—conflicts with the structural requirements of the grammar. The key scenario involves *bridge events*: events that span temporal gaps between clusters (bursts) of activity. These bridge events typically have low weight (they are "filler" material, not high-salience focal events), so the weight-maximising greedy policy skips them. But without bridge events, the selected set may violate *span constraints*—requirements that the narrative arc covers a minimum fraction of the simulation timeline.

Example 4.3 (Class C failure)**.** A valid arc requires events spanning at least 15% of the simulation timeline. The greedy solver, maximising total weight, selects high-weight events clustered in a short interval (say, positions 0.40–0.55, covering only 15%). But the grammar also requires $k = 3$ development events before the turning point, and the highest-weight development candidates all fall in the same cluster. The solver has committed to a dense, high-weight event set that satisfies the weight objective but violates the span constraint.

A TP-conditioned solver that first fixes the turning point and then optimises subject to span requirements avoids this failure by selecting bridge events that maintain temporal coverage, even at a cost in total weight.

Class C failures account for 18 of 138 failures (13.0%). Recovery rates across different constraint sets are 17/20 (85%) and 21/26 (80.8%), reflecting the difficulty of balancing weight maximisation against temporal coverage.

4.1.4 Class D: Assembly Compression Failures

Class D failures arise from a structural imbalance in the assembled narrative arc. The grammar is satisfied—the DFA accepts, and the prefix requirement is met—but the phase structure is dramatically lopsided. Weight-maximising event selection concentrates high-weight events in the RESOLUTION phase (post-turning-point), starving the DEVELOPMENT phase of material.

Example 4.4 (Class D failure)**.** Among 20 selected events, 15 have higher weight than the turning point and occur after it in the timeline. Only 1 event is assigned to SETUP and 2 to DEVELOPMENT, barely meeting the prefix requirement of $k = 3$. The grammar is technically satisfied, but the resulting arc is hollow: the narrative jumps almost immediately from setup to resolution with minimal development.

Span-VAG (viability-aware greedy with span filtering) addresses this by filtering candidate events for span viability before weight maximisation. Events that would compress the phase structure below a span threshold are excluded from the candidate pool. Recovery rate: 55/57 (96.5%).

Class D is the second most prevalent class at 57 of 138 failures (41.3%), but it is also the most tractable: the span-VAG fix recovers nearly all instances.

Algorithm 1 Viability-Aware Greedy Extraction (VAG)

Require: Pool P, anchor e_0, grammar $\mathcal{A}$ with prefix requirement k, max length $n_{\max}$, viability mode MODE $\in \{\text{SPAN}, \text{GAP}, \text{BUDGET}\}$
Ensure: Selected event sequence S (sorted by timestamp)

function VAG_EXTRACT(P, e_0, $\mathcal{A}$, $n_{\max}$, MODE)
  $S \leftarrow \{e_0\}$ ▷ Seed with anchor event
  $R \leftarrow \text{SORTBYWEIGHT}(P \setminus \{e_0\})$ ▷ Remaining candidates, weight-descending
  while $|S| < n_{\max}$ **and** $R \neq \varnothing$ **do**
    accepted $\leftarrow$ FALSE
    for each $c \in R$ in weight order **do**
      if ISVIABLE(S, c, $R \setminus \{c\}$, $\mathcal{A}$, $n_{\max}$, MODE) **then**
        $S \leftarrow S \cup \{c\}$; $R \leftarrow R \setminus \{c\}$
        accepted $\leftarrow$ TRUE; **break**
      end if
    end for
    if $\neg$ *accepted* **then**
      break ▷ No viable candidate remains
    end if
  end while
  return SORTBYTIMESTAMP(S)
end function

4.1.5 Viability-Aware Greedy (VAG) Algorithms

The VAG family of algorithms shares a common skeleton: greedily select events in weight-descending order, but reject any candidate whose addition would render a future valid completion impossible. The variants differ only in which viability checks are applied. Algorithm 1 presents the common framework; Algorithm 2 details the viability predicate with the three filtering levels.

The three modes correspond to the hierarchy levels: **Span-VAG** (Level 1) uses only the common checks, **Gap-VAG** (Level 2) adds the gap-bridgeability check, and **BVAG** (Level 2.5) further adds the bridge-budget reservation that ensures the remaining slot count can accommodate the minimum number of bridge events implied by oversized gaps.

Remark 4.5 (Failure classes are not mutually exclusive). A single failing instance may exhibit multiple failure classes simultaneously. For example, a Class B misclassification can reduce j_{dev} to the point where a Class A absorbing state is triggered. Similarly, the assembly compression of Class D often co-occurs with the commitment timing issues of Class C. The prevalence figures in Table 4.1 count each failure under its *primary* class—the earliest stage at which a fix would have prevented the failure—and therefore sum to more than 100% of unique failures.

4.2 Constraint Antagonism

The most surprising result in the failure analysis is not that different algorithms fix different failure classes—that is expected. The surprise is that fixes for one class can *worsen* another.

The Span-VAG paradox. Span-VAG achieves 62.7% validity on bursty instances with gap constraints, a substantial improvement over the myopic greedy baseline of 54.7%. Yet on *multi-burst* instances with gap constraints, Span-VAG achieves 0% validity—strictly *worse* than the myopic greedy policy's 8%.

The mechanism is as follows. Span viability favours selecting events at burst endpoints to maximise temporal coverage of the narrative arc. In multi-burst topologies, temporal valleys separate the bursts. The low-weight bridge events in these valleys are essential for gap feasibility: without them, the selected events have temporal gaps that violate constraints. But Span-VAG, by favouring endpoint events for their span contribution, systematically skips these low-weight bridge events. The result is a selected set that has excellent span coverage but catastrophic gap violations.

Partial order, not total order. This antagonism means that the algorithm hierarchy is a *partial order*, not a total order. No single algorithm dominates across all constraint combinations. Span-VAG dominates greedy on bursty+gap but is dominated by greedy on multi-burst+gap. Gap-VAG dominates greedy on both topologies but is incomparable with Span-VAG (better on multi-burst+gap, different on bursty+gap).

Figure 4.1 illustrates this partial order. The directed edges indicate strict dominance: algorithm X dominates algorithm Y if X achieves weakly higher validity than Y on every constraint combination and strictly higher on at least one. The key feature is the pair of incomparable nodes—Span-VAG and Gap-VAG—which cannot be ordered. Both feed into BVAG (budget-aware VAG), which resolves the antagonism by jointly optimising span and gap constraints. The full hierarchy culminates in the TP-Solver and, ultimately, the oracle.

4.3 The Constructive Hierarchy

Table 4.2 presents the full algorithm hierarchy. Each level introduces one new capability that addresses a specific failure class (or combination of classes) that the previous levels cannot handle.

We now describe each level, what it fixes, and what it cannot fix.

Level 0: Myopic Greedy. The baseline policy selects events in decreasing weight order, assigns phase labels greedily, and picks the turning point endogenously as $e^* = \arg\max_{v:a(v)=a^*} w(v)$. It has no lookahead and no constraint awareness beyond the DFA itself. It fails on all four classes: Class A by selecting a bad pivot, Class B by using a position-based classifier, Class C by ignoring commitment timing, and Class D by concentrating weight in the post-TP region.

Level 1: Span-VAG. Adds a viability filter that excludes candidate events whose inclusion would compress the arc's temporal span below a threshold. This addresses Class D failures (assembly compression) by ensuring that the selected events maintain adequate temporal coverage. However, the span filter can antagonise gap constraints, producing the 0% validity on multi-burst+gap documented in Section 4.2.

Level 2: Gap-VAG. Replaces the span-only viability filter with a gap-aware filter that checks both span and maximum temporal gap constraints. This resolves the Span-VAG antagonism on multi-burst topologies, lifting validity from 0% to 60% on multi-burst+gap. Gap-VAG also improves on Span-VAG for bursty+gap (81.3% vs. 62.7%), making it the first algorithm in the hierarchy that is not dominated on any tested constraint combination.

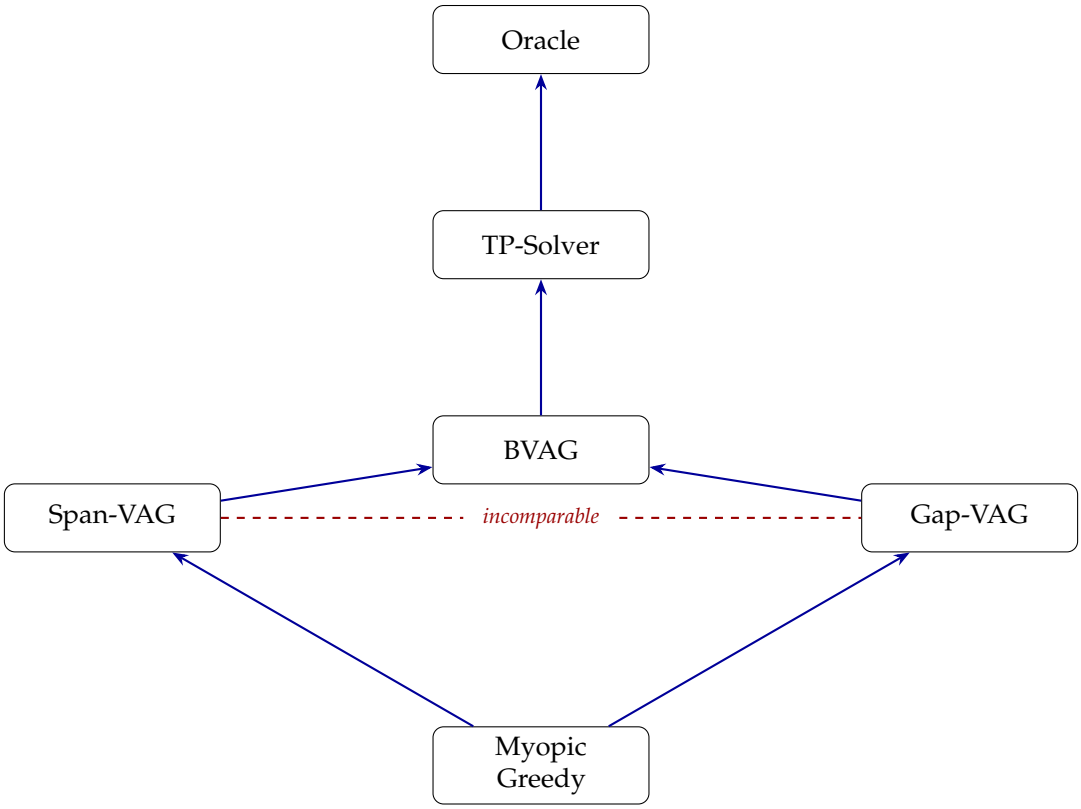


Figure 4.1: Partial order on solver algorithms. Solid arrows indicate strict dominance (higher validity on every constraint combination). The dashed line between Span-VAG and Gap-VAG indicates incomparability: neither dominates the other. BVAG resolves the antagonism; the TP-Solver and Oracle complete the hierarchy.

Level 2.5: BVAG (Budget-Aware VAG). Augments Gap-VAG with a budget constraint that limits the number of events allocated to each phase. This provides a marginal improvement (82.0% vs. 81.3% on bursty+gap) by preventing extreme phase imbalance. The gains are modest because Gap-VAG already handles most assembly compression; the budget constraint catches the residual cases.

Level 3: TP-Solver ($M = 25$). The first algorithm in the hierarchy to use an *outer loop* over turning-point candidates. Instead of accepting the endogenous arg max pivot [7], the TP-Solver considers the top-M focal events by weight and, for each candidate, solves the inner problem (find the best valid event selection given a fixed TP) via label-setting dynamic programming. This outer loop directly attacks Class A failures: if the highest-weight pivot yields $j_{\text{dev}} < k$, the solver tries the next candidate, and the next, until a feasible pivot is found.

The TP-Solver also mitigates Class C failures because fixing the TP *Markovises* the inner problem: once the pivot is known, every event's phase label is determined by its temporal relationship to the fixed TP. The inner problem reduces to a resource-constrained shortest path problem [12] (RCSPP), which the label-setting DP solves efficiently with dominance pruning. Details appear in Section 4.4.

Level ∞: Oracle. The oracle enumerates all feasible subsets and selects the one with maximum total weight. It serves as the theoretical upper bound: any valid sequence that exists will be found.

Table 4.2: Constructive algorithm hierarchy. Validity rates are measured on the bursty+gap and multi-burst+gap constraint combinations. Complexity expressions use n for pool size, M for the number of TP candidates, L for label-setting queue size, d for DP state dimensions, q for budget states, and S for the number of feasible subsets enumerated by the oracle.

Level	Algorithm	Description	TP Selection	B+G	MB+G	Complexity
0	Myopic Greedy	Weight-max, no lookahead	Endogenous	54.7%	8%	$O(n^2)$
1	Span-VAG	Viability filter on span	Endogenous	62.7%	0%	$O(n^2)$
2	Gap-VAG	Gap-aware viability	Endogenous	81.3%	60%	$O(n^2 \cdot pool)$
2.5	BVAG	Budget-aware VAG	Endogenous	82.0%	60%	$O(n^2 \cdot q)$
3	TP-Solver	DP over TP candidates	Outer loop	96.0%	94%	$O(M \cdot L \cdot n^2 \cdot d)$
∞	Oracle	Exhaustive enumeration	All focal	99.3%	100%	$O(n^2 \cdot S)$

The oracle achieves 99.3% validity on bursty+gap (the residual 0.7% represents instances where no valid sequence exists under any pivot assignment) and 100% on multi-burst+gap.

Remark 4.6 (The oracle gap). The gap between the TP-Solver (96.0%) and the oracle (99.3%) on bursty+gap is 3.3 percentage points. This gap represents instances where: (i) the correct pivot is not among the top-$M = 25$ candidates, or (ii) the label-setting DP's dominance criterion prunes the optimal solution. Increasing M narrows the gap at the cost of linear growth in runtime. The 94%–100% gap on multi-burst+gap has a similar origin.

4.4 The TP-Conditioned Solver

The TP-conditioned solver is the practically important algorithm in the hierarchy: it achieves near-oracle validity at polynomial cost. Algorithm 3 presents the pseudocode; the remainder of this section provides a line-by-line explanation.

Line-by-line explanation.

Line 2: Candidate extraction. The solver selects the top-M focal events by weight as turning-point candidates. This is the outer loop's search space. Setting $M = 25$ captures the vast majority of viable pivots: in practice, the correct pivot is almost always among the 25 highest-weight focal events.

Line 3: Initialisation. The best solution found so far is empty, with score $-\infty$.

Line 4: Outer loop over candidates. For each candidate c, the solver fixes c as the turning point and invokes the inner DP. This is the key structural move: by fixing the TP, the solver eliminates the endogenous coupling that makes the full problem hard.

Line 5: Markovisation. Fixing c as the turning point *Markovises* the inner problem. The reason is that, once the pivot is known, every event's phase label is determined by a simple rule: events before c in the timeline are candidates for SETUP or DEVELOPMENT; c itself is TURNING_POINT; events after c are candidates for RESOLUTION. The phase classifier becomes *deterministic*—no longer dependent on the DFA's state trajectory. The inner problem therefore reduces to a Resource-Constrained Shortest Path Problem (RCSPP): find the weight-maximising subset of events that satisfies the grammar's prefix requirement ($\geq k$ development events before c) and any additional span or gap constraints.

Line 6: Label-Setting DP. The inner solver is a label-setting dynamic program that explores the state space of partial event selections. Each state is a tuple encoding the current score, the number of phase slots used, and the earliest timestamp in the selection. The DP propagates labels (partial solutions) forward through the event pool, pruning dominated labels at each step.

The dominance criterion is: label A dominates label B if and only if

(i) $A.score \geq B.score$,

(ii) $A.slots\_used \leq B.slots\_used$, and

(iii) $A.first\_time \leq B.first\_time$.

This three-dimensional dominance criterion prunes the search space dramatically. A label that is worse on score, uses more phase slots, and starts later in the timeline can never lead to a solution that the dominating label could not also produce.

Lines 7–9: Solution update. If the inner DP returns a valid solution with a higher score than the current best, the solver updates its record.

Line 12: Return. After exhausting all M candidates, the solver returns the best valid solution found, or $\varnothing$ if no candidate yielded a valid selection.

Remark 4.7 (Why fixing the TP Markovises the problem). Under the greedy policy, the phase label assigned to an event depends on the DFA's state at the moment the event is processed, which in turn depends on all previously assigned labels. This creates a sequential dependency that prevents decomposition.

Fixing the TP breaks this dependency. With the pivot c known, the timeline splits into three regions: pre-c, the event c itself, and post-c. Events in the pre-c region compete for SETUP and DEVELOPMENT slots; events in the post-c region are assigned RESOLUTION. The assignment of labels in the pre-c region is independent of the post-c region (and vice versa), conditioned on the fixed TP. This conditional independence is precisely the Markov property: the future (post-TP) is independent of the past (pre-TP) given the present (the TP itself).

Remark 4.8 (Complexity). The outer loop runs M iterations. Each iteration invokes the label-setting DP, which processes n events, maintaining a label set of size at most L at each event. Each label has d state dimensions. The total work is $O(M \cdot L \cdot n^2 \cdot d)$. In practice, dominance pruning keeps L small (typically $L \ll n$), and $d = 3$ (score, slots used, first time), making the algorithm efficient for event pools of moderate size.

4.5 Failure Decomposition

The four-class taxonomy induces a natural decomposition of the overall failure probability.

Table 4.3: Approximation quality ratios (solver score divided by oracle score) for instances where both solver and oracle find valid sequences.

Topology	Mean ratio	Minimum ratio
Bursty	0.996	0.975
Multi-burst	0.9905	0.9512

Proposition 4.9 (Failure decomposition)**.** *Let $P(\text{fail})$ denote the probability that a greedy policy fails to produce a valid sequence. Then*

$$P(\text{fail}) \;=\; P(j_{\text{dev}} < k) \;+\; \big[1 - P(j_{\text{dev}} < k)\big] \;\cdot\; P\big(\text{other\_fail} \;\big|\; j_{\text{dev}} \geq k\big). \tag{4.1}$$

Proof. The law of total probability, partitioning on whether the absorbing-state condition $j_{\text{dev}} < k$ holds, gives

$$P(\text{fail}) = P(\text{fail} \mid j_{\text{dev}} < k)\, P(j_{\text{dev}} < k) + P(\text{fail} \mid j_{\text{dev}} \geq k)\, P(j_{\text{dev}} \geq k).$$

By Theorem 3.8, $P(\text{fail} \mid j_{\text{dev}} < k) = 1$: every instance with $j_{\text{dev}} < k$ fails. Substituting and writing $P(\text{other\_fail} \mid j_{\text{dev}} \geq k)$ for the residual failure rate yields Equation (4.1). □

The decomposition has a clean algorithmic interpretation. Each level of the constructive hierarchy attacks a different term:

- **First term:** $P(j_{\text{dev}} < k)$**.** These are Class A (absorbing state) failures. They are predicted exactly by Theorem 3.8 and can only be attacked by changing the pivot—the strategy of the TP-Solver's outer loop.
- **Second term: residual failures given** $j_{\text{dev}} \geq k$**.** These are Classes B, C, and D, conditional on escaping absorption. The grammar-aware classifier eliminates Class B entirely. Span-VAG and Gap-VAG attack Class D. The TP-Solver's inner DP, by Markovising the label-assignment problem, attacks Class C.

Remark 4.10 (Bridge to the absorbing ideal)*.* The first term in the failure decomposition is precisely the measure of the absorbing ideal of Chapter 7: $P(j_{\text{dev}} < k)$ is the probability that the greedy policy produces a context element in the absorbing set $\perp$. The decomposition thus connects the algebraic theory (the ideal is inescapable under commitment) to the empirical hierarchy (the TP-Solver attacks the ideal by exploring alternative pivots).

4.6 Approximation Quality

When a valid sequence exists and is found by the solver, how close is its quality to the optimum? The answer is: remarkably close.

Table 4.3 reports the approximation quality ratios (solver score divided by oracle score) for instances where both the TP-Solver and the oracle find valid sequences. On bursty topologies, the mean ratio is 0.996 with a worst case of 0.975: the solver typically achieves within 0.4% of optimal, and never worse than 2.5%. On multi-burst topologies, the mean ratio is 0.9905 with a worst case of 0.9512: within 1% on average and within 5% in the worst case.

Remark 4.11 (Feasibility-dominated landscape). These near-optimal quality ratios reveal a fundamental feature of the problem landscape: it is *feasibility-dominated*. The primary challenge is finding *any* valid solution, not optimising among many. Once a valid solution exists, its quality is typically within 1–5% of optimal.

This observation justifies the focus on structural feasibility throughout the constructive hierarchy. The hierarchy's levels are ordered by their ability to *find valid solutions*, not by their ability to optimise among valid solutions. The approximation quality data confirm that feasibility is the binding constraint: solve the feasibility problem, and the optimisation problem largely solves itself.

Remark 4.12 (Implications for algorithm design). The feasibility-dominated landscape has a practical implication for algorithm design: it is more profitable to invest computational budget in exploring additional TP candidates (increasing M) than in refining the inner solver's optimisation. Each additional TP candidate opens a new region of the feasibility space; a better inner solver merely polishes the score within a region that has already been found. The marginal value of an extra candidate exceeds the marginal value of a tighter approximation.

4.7 Exercises

Exercise 4.1 (Classifying failures)**.** Consider an event graph with $n = 15$, $k = 3$, and a greedy turning point e^* at temporal position 0.6 (normalised). The greedy solver selects 12 events, of which 9 are in the post-TP region. The grammar is satisfied, but the arc's temporal span covers only 20% of the timeline.

(a) Which failure class does this instance primarily belong to?

(b) Which secondary class may also apply? Explain.

(c) Which level of the constructive hierarchy is the minimum required to fix this failure?

Hint: The grammar is satisfied (ruling out Class A and B), but the phase structure is lopsided (Class D) and the temporal span is narrow (which may also indicate Class C commitment timing issues).

Exercise 4.2 (Antagonism construction)**.** Construct a concrete multi-burst event graph (with two bursts separated by a valley) in which:

(a) Span-VAG selects events at the endpoints of each burst, achieving good span coverage but violating a gap constraint (maximum gap ≤ 0.15 of the timeline).

(b) Myopic greedy, by not filtering for span, happens to include a low-weight bridge event in the valley, satisfying the gap constraint.

Verify that Span-VAG fails while greedy succeeds on this instance.

Exercise 4.3 (Markovisation)**.** Let $P = \{e_1, \ldots, e_8\}$ be an event pool with $k = 2$. Suppose the greedy policy selects e_3 as the turning point.

(a) Write down the three regions induced by fixing e_3 as the TP: which events are pre-TP, which is the TP, and which are post-TP?

(b) Explain why the label assignment in the pre-TP region is independent of the post-TP region, given the fixed TP.

(c) Suppose e_5 has a higher weight than e_3. Under endogenous TP selection, e_5 would be the pivot. Explain why the TP-Solver can still consider e_3 and why this is valuable.

Exercise 4.4 (Failure decomposition with real numbers)**.** In a domain where $P(j_{\text{dev}} < k) = 0.136$ (Class A) and the residual failure rate given $j_{\text{dev}} \geq k$ is $P(\text{other\_fail} \mid j_{\text{dev}} \geq k) = 0.52$:

(a) Compute the overall failure probability using Equation (4.1).

(b) A grammar-aware classifier eliminates all Class B failures, reducing the residual failure rate to 0.12. Compute the new overall failure probability.

(c) The TP-Solver eliminates 90% of Class A failures (by finding alternative pivots). If we also use the grammar-aware classifier, what is the overall failure probability?

Solution sketch for (a): $P(\text{fail}) = 0.136 + (1 - 0.136) \cdot 0.52 = 0.136 + 0.864 \cdot 0.52 = 0.136 + 0.449 = 0.585$.

Exercise 4.5 (Oracle gap analysis)**.** The TP-Solver with $M = 25$ achieves 96.0% validity on bursty+gap, while the oracle achieves 99.3%. The gap is 3.3%.

(a) If the correct pivot is uniformly distributed among all focal events and there are 50 focal events, what is the probability that the correct pivot is *not* among the top-25 by weight? (Assume the correct pivot is equally likely to be any focal event.)

(b) In practice, the correct pivot is highly correlated with weight. If 95% of correct pivots are among the top-25 by weight, and the inner DP recovers 98% of these, what validity rate does the model predict?

(c) How does this compare with the observed 96.0%?

Algorithm 2 Viability Predicate (IsViable)

Require: Current selection S, candidate c, remaining pool R, grammar $\mathcal{A}$, max length $n_{\max}$, mode $\in \{\text{SPAN}, \text{GAP}, \text{BUDGET}\}$

Ensure: TRUE if adding c preserves feasibility

function IsViable(S, c, R, $\mathcal{A}$, $n_{\max}$, MODE)
  $S' \leftarrow S \cup \{c\}$; $r \leftarrow n_{\max} - |S'|$ ▷ Remaining slots
  ▷ **— Common checks (all modes) —**
  if $|S'| > n_{\max}$ **then return** FALSE ▷ Length exceeded
  end if
  if $|S'| + \min(r, |R|) < n_{\min}$ **then return** FALSE ▷ Cannot reach min length
  end if
  if MaxCoverage$(S', R) < \rho$ **then return** FALSE ▷ Focal-actor coverage unreachable
  end if
  if MaxSpan$(S', R, r) < \gamma \cdot T$ **then return** FALSE ▷ Span unreachable
  end if
  if TPLocked$(S', R, \mathcal{A})$ **then return** FALSE ▷ TP locked with $< k$ dev events
  end if
  ▷ **— Gap-aware check (modes: gap, budget) —**
  if MODE $\in \{\text{GAP}, \text{BUDGET}\}$ **then**
    for each oversized gap (t_l, t_r) in S' with $t_r - t_l > G_{\max}$ **do**
      if no bridge event exists in R with $t_l < t(v) < t_r$ **then return** FALSE ▷ Gap unbridgeable
      end if
    end for
  end if
  ▷ **— Budget-aware check (mode: budget) —**
  if MODE = BUDGET **then**
    $B_{\text{lb}} \leftarrow \sum_{(t_i, t_{i+1}) \in S'} \max\big(0, \lceil (t_{i+1} - t_i)/G_{\max} \rceil - 1\big)$
    if $B_{\text{lb}} > r$ **then return** FALSE ▷ Bridge budget exceeded
    end if
  end if
  return TRUE
end function

Algorithm 3 TP-Conditioned Solver

Require: Event pool P, prefix requirement k, candidate count M
Ensure: Best valid event selection, or $\varnothing$ if none found

```
function TP_SOLVER(P, k, M)
    candidates ← top-M focal events by weight from P
    best_solution ← ∅;   best_score ← −∞
    for each candidate c in candidates do
                                  ▷ Fix c as TURNING_POINT—this Markovises the inner problem
        solution ← LABELSETTINGDP(P, c, k)
        if solution is valid and SCORE(solution) > best_score then
            best_solution ← solution
            best_score ← SCORE(solution)
        end if
    end for
    return best_solution
end function
```

Chapter 5

Context Algebra

This chapter develops the algebraic foundation on which the rest of the theory is built [5]. We introduce the *context element*—a compact summary of the structural state of a block of events—and define an associative composition operator that merges adjacent blocks while faithfully tracking the position of the global pivot. The resulting monoid is the engine that powers both the streaming left-fold implementation and the parallel holographic tree.

5.1 The Context Element

Every contiguous block of events can be compressed into a triple that records three structural quantities: where the strongest focal event sits, how much development capacity the block contains in total, and how much of that capacity falls before the strongest focal event. These are the only three numbers required to compose blocks correctly.

Definition 5.1 (Context element)**.** A **context element** is a triple

$$C \;=\; \big(\, w^\star,\; d_{\text{total}},\; d_{\text{pre}} \,\big),$$

where the components are defined as follows.

(i) $w^\star \in \mathbb{R} \cup \{-\infty\}$ is the **local pivot key**: the maximum weight among all focal-actor events in the block. If the block contains no focal events, $w^\star = -\infty$.

(ii) $d_{\text{total}} \in \mathbb{N}$ is the **total development capacity**: the count of non-focal events in the block. Each non-focal event contributes one unit of development capacity regardless of its weight or timestamp.

(iii) $d_{\text{pre}} \in \mathbb{N}$ is the **pre-pivot development capacity**: the count of non-focal events whose timestamp strictly precedes the timestamp of the local pivot (the focal event attaining $w^\star$). When $w^\star = -\infty$ (no focal events), d_{pre} is defined to be zero.

The intuition behind each component is direct:

- **Pivot key $w^\star$.** The pivot is the focal event that the endogenous selection mechanism has chosen—the strongest signal in the block. Its weight $w^\star$ determines whether this block's pivot can survive composition with adjacent blocks: if a neighbour contains a focal event with a strictly larger weight, the global pivot will shift to that neighbour.

- **Total development d_{total}.** The total count of non-focal events measures the raw amount of development capacity that the block contributes. Under composition this quantity simply adds, since every non-focal event remains non-focal regardless of where the pivot falls.

- **Pre-pivot development d_{pre}.** This is the structurally subtle component. The feasibility condition for the endogenous pivot (typically requiring at least k non-focal events before the pivot) depends on how many non-focal events appear *before* the global pivot in the combined sequence. The quantity d_{pre} records this count relative to the block's own local pivot, but as we shall see in Section 5.2, composition may promote additional events into the pre-pivot region when the global pivot shifts rightward.

Example 5.2 (Concrete context element)**.** Consider a block of seven events arranged in temporal order:

$$\underbrace{e_1}_{\text{non-focal}},\ \underbrace{e_2}_{\substack{\text{focal}\\ w=3}},\ \underbrace{e_3}_{\text{non-focal}},\ \underbrace{e_4}_{\text{non-focal}},\ \underbrace{e_5}_{\substack{\text{focal}\\ w=7}},\ \underbrace{e_6}_{\text{non-focal}},\ \underbrace{e_7}_{\text{non-focal}}.$$

The focal events are e_2 (weight 3) and e_5 (weight 7). The local pivot is e_5 since $7 > 3$, giving $w^\star = 7$.

The non-focal events are e_1, e_3, e_4, e_6, e_7, so $d_{\text{total}} = 5$.

Among those five non-focal events, the ones with timestamps strictly preceding e_5 are e_1, e_3, e_4—three events—giving $d_{\text{pre}} = 3$.

Therefore the context element for this block is

$$C = (7,\ 5,\ 3).$$

5.2 Endogenous Composition

We now define the binary operation that composes two adjacent context elements—the left block C_A followed in time by the right block C_B—into a single context element representing the concatenated sequence.

Definition 5.3 (Endogenous composition $\otimes_{\text{endo}}$)**.** Let $C_A = (w^\star{}_A,\ d_{\text{total}\,A},\ d_{\text{pre},A})$ and $C_B = (w^\star{}_B,\ d_{\text{total}\,B},\ d_{\text{pre},B})$ be context elements. Their **endogenous composition** is

$$C_A \otimes_{\text{endo}} C_B \ = \ \big(\, w^\star,\ d_{\text{total}},\ d_{\text{pre}}\,\big),$$

where

$$w^\star = \max\big(w^\star{}_A,\ w^\star{}_B\big), \tag{5.1}$$

$$d_{\text{total}} = d_{\text{total}\,A} + d_{\text{total}\,B}, \tag{5.2}$$

$$d_{\text{pre}} = \begin{cases} d_{\text{pre},A} & \text{if } w^\star{}_A \geq w^\star{}_B, \\ d_{\text{total}\,A} + d_{\text{pre},B} & \text{if } w^\star{}_B > w^\star{}_A. \end{cases} \tag{5.3}$$

The rules for $w^\star$ and d_{total} are immediate: the global pivot key is the maximum over both blocks, and total development capacity is additive. The rule for d_{pre} requires more careful thought.

Remark 5.4 (The pivot-shift insight). The d_{pre} rule in Equation (5.3) encodes the following critical structural insight. There are two regimes:

1. **Pivot stays in the left block** ($w^\star{}_A \geq w^\star{}_B$). The global pivot is the same event that was the local pivot of block A. The non-focal events before the global pivot are exactly those that were before A's local pivot. Nothing in block B can contribute to the pre-pivot count because the entire right block sits *after* the pivot. Hence $d_{\text{pre}} = d_{\text{pre},A}$.

2. **Pivot shifts to the right block** ($w^\star{}_B > w^\star{}_A$). The global pivot now lives inside block B. Every event in block A—focal or non-focal—has a timestamp that precedes the global pivot. In particular, *all* $d_{\text{total}\,A}$ non-focal events in block A are now in the pre-pivot region. Within block B itself, the events before B's local pivot are counted by $d_{\text{pre},B}$. Therefore the total pre-pivot development count is $d_{\text{total}\,A} + d_{\text{pre},B}$.

This asymmetry—the left block's *entire* development capacity gets promoted upon a rightward pivot shift, but the right block contributes nothing additional when the pivot stays left—is the source of non-commutativity (see Section 5.5).

Example 5.5 (Pivot shifts right)**.** Let $A = (3, 5, 2)$ and $B = (7, 4, 1)$. Since $w^\star{}_B = 7 > w^\star{}_A = 3$, the pivot shifts to block B. Applying Definition 5.3:

$$\begin{aligned} w^\star &= \max(3, 7) = 7, \\ d_{\text{total}} &= 5 + 4 = 9, \\ d_{\text{pre}} &= d_{\text{total}\,A} + d_{\text{pre},B} = 5 + 1 = 6. \end{aligned}$$

Therefore $A \otimes_{\text{endo}} B = (7, 9, 6)$.

The interpretation: all five of A's non-focal events now sit before the global pivot (which is inside B), contributing 5 to the pre-pivot count. Within B, only one non-focal event precedes B's local pivot, contributing 1. The total pre-pivot development capacity is 6.

Example 5.6 (Pivot stays left)**.** Now let $A = (7, 4, 1)$ and $B = (3, 5, 2)$. Since $w^\star{}_A = 7 \geq w^\star{}_B = 3$, the pivot stays in block A:

$$\begin{aligned} w^\star &= \max(7, 3) = 7, \\ d_{\text{total}} &= 4 + 5 = 9, \\ d_{\text{pre}} &= d_{\text{pre},A} = 1. \end{aligned}$$

Therefore $A \otimes_{\text{endo}} B = (7, 9, 1)$.

Here the global pivot remains in A, where only one non-focal event preceded the local pivot. The entire right block B lies after the pivot, so none of B's events contribute to d_{pre}.

5.3 Associativity

The central algebraic property of $\otimes_{\text{endo}}$ is associativity: we may parenthesise a three-way composition in either order and obtain the same result. This is what allows us to decompose a long event sequence into arbitrary contiguous blocks, compose each block independently, and then combine them—the hallmark of a parallel-friendly reduction.

Proposition 5.7 (Associativity of $\otimes_{\text{endo}}$)**.** *For any context elements A, B, and C,*

$$(A \otimes_{\text{endo}} B) \otimes_{\text{endo}} C \;=\; A \otimes_{\text{endo}} (B \otimes_{\text{endo}} C).$$

Proof. Write $A = (w^\star{}_A, d_A, p_A)$, $B = (w^\star{}_B, d_B, p_B)$, and $C = (w^\star{}_C, d_C, p_C)$, where we abbreviate d_{total} and d_{pre} as d and p for readability.

The $w^\star$ component of both sides equals $\max(w^\star{}_A, w^\star{}_B, w^\star{}_C)$ by associativity of max. The d_{total} component of both sides equals $d_A + d_B + d_C$ by associativity of addition. It remains to verify the d_{pre} component.

We proceed by exhaustive case analysis on which block contains the global pivot.

Case 1: Global pivot in A ($w^\star{}_A \geq w^\star{}_B$ and $w^\star{}_A \geq w^\star{}_C$).

Left-associated: $(A \otimes_{\text{endo}} B) \otimes_{\text{endo}} C$. Let $AB = A \otimes_{\text{endo}} B$. Since $w^\star{}_A \geq w^\star{}_B$, we have $AB = (w^\star{}_A,\ d_A + d_B,\ p_A)$. Then $AB \otimes_{\text{endo}} C$: since $w^\star{}_A \geq w^\star{}_C$, the result is $(w^\star{}_A,\ d_A + d_B + d_C,\ p_A)$.

Right-associated: $A \otimes_{\text{endo}} (B \otimes_{\text{endo}} C)$. Let $BC = B \otimes_{\text{endo}} C$. Two subcases arise:

(a) $w^\star{}_B \geq w^\star{}_C$: $BC = (w^\star{}_B,\ d_B + d_C,\ p_B)$. Then $A \otimes_{\text{endo}} BC$: since $w^\star{}_A \geq w^\star{}_B$, the result is $(w^\star{}_A,\ d_A + d_B + d_C,\ p_A)$. ✓

(b) $w^\star{}_C > w^\star{}_B$: $BC = (w^\star{}_C,\ d_B + d_C,\ d_B + p_C)$. Then $A \otimes_{\text{endo}} BC$: since $w^\star{}_A \geq w^\star{}_C$, the result is $(w^\star{}_A,\ d_A + d_B + d_C,\ p_A)$. ✓

In both subcases the d_{pre} component is p_A, matching the left-associated result.

Case 2: Global pivot in B ($w^\star{}_B > w^\star{}_A$ and $w^\star{}_B \geq w^\star{}_C$).

Left-associated: $(A \otimes_{\text{endo}} B) \otimes_{\text{endo}} C$. Since $w^\star{}_B > w^\star{}_A$, $AB = (w^\star{}_B,\ d_A + d_B,\ d_A + p_B)$. Then $AB \otimes_{\text{endo}} C$: since $w^\star{}_B \geq w^\star{}_C$, the result is $(w^\star{}_B,\ d_A + d_B + d_C,\ d_A + p_B)$.

Right-associated: $A \otimes_{\text{endo}} (B \otimes_{\text{endo}} C)$. Since $w^\star{}_B \geq w^\star{}_C$, $BC = (w^\star{}_B,\ d_B + d_C,\ p_B)$. Then $A \otimes_{\text{endo}} BC$: since $w^\star{}_B > w^\star{}_A$, the result is $(w^\star{}_B,\ d_A + d_B + d_C,\ d_A + p_B)$. ✓

Key insight (why Case 2 is subtle). The quantity d_A appears in d_{pre} on both sides because it gets "promoted" into the pre-pivot region exactly once—when the global pivot moves from the left sub-expression into block B. In the left-associated computation, this promotion happens during the $A \otimes_{\text{endo}} B$ step. In the right-associated computation, it happens during the $A \otimes_{\text{endo}} BC$ step. Either way, d_A is promoted exactly once and combined with p_B, yielding the same final $d_{\text{pre}} = d_A + p_B$. The single-promotion invariant is what makes associativity non-obvious here, and it is the reason the operator is *not* commutative (see Section 5.5): the direction of the pivot shift determines whether promotion occurs.

Case 3: Global pivot in C ($w^\star{}_C > w^\star{}_A$ and $w^\star{}_C > w^\star{}_B$).

Left-associated: $(A \otimes_{\text{endo}} B) \otimes_{\text{endo}} C$. First form $AB = A \otimes_{\text{endo}} B$. Regardless of whether $w^\star{}_A \geq w^\star{}_B$ or $w^\star{}_B > w^\star{}_A$, the total development of AB is $d_{AB} = d_A + d_B$. Then $AB \otimes_{\text{endo}} C$: since $w^\star{}_C > w^\star{}_{AB}$, the pivot shifts right, giving

$$d_{\text{pre}} = d_{AB} + p_C = d_A + d_B + p_C.$$

The result is $(w^\star{}_C,\ d_A + d_B + d_C,\ d_A + d_B + p_C)$.

Right-associated: $A \otimes_{\text{endo}} (B \otimes_{\text{endo}} C)$. Since $w^\star{}_C > w^\star{}_B$, $BC = (w^\star{}_C,\ d_B + d_C,\ d_B + p_C)$. Then $A \otimes_{\text{endo}} BC$: since $w^\star{}_C > w^\star{}_A$, the pivot shifts right, giving

$$d_{\text{pre}} = d_A + (d_B + p_C) = d_A + d_B + p_C.$$

The result is $(w^\star{}_C,\ d_A + d_B + d_C,\ d_A + d_B + p_C)$. ✓

All three cases yield identical results under both parenthesisations. Since every possible ordering of $w^\star{}_A, w^\star{}_B, w^\star{}_C$ falls into exactly one of these cases (with ties handled by the $\geq$ / $>$ structure), the proof is complete. □

We consolidate the result with a numerical verification of the subtle Case 2.

Example 5.8 (Numerical verification of Case 2)**.** Let $A = (3,5,2)$, $B = (7,4,1)$, $C = (2,3,1)$. The global pivot is in B since ${w^\star}_B = 7 > {w^\star}_A = 3$ and ${w^\star}_B = 7 \geq {w^\star}_C = 2$.

Left-associated. $AB = A \otimes_{\text{endo}} B$: since $7 > 3$, we get $AB = (7,\ 5+4,\ 5+1) = (7,9,6)$. Then $AB \otimes_{\text{endo}} C$: since $7 \geq 2$, the result is $(7,\ 9+3,\ 6) = (7,12,6)$.

Right-associated. $BC = B \otimes_{\text{endo}} C$: since $7 \geq 2$, we get $BC = (7,\ 4+3,\ 1) = (7,7,1)$. Then $A \otimes_{\text{endo}} BC$: since $7 > 3$, the result is $(7,\ 5+7,\ 5+1) = (7,12,6)$.

Both sides yield $(7,12,6)$, confirming associativity. □

5.4 Identity Element

Proposition 5.9 (Identity element for $\otimes_{\text{endo}}$)**.** *The element*

$$e \ = \ (-\infty,\ 0,\ 0)$$

is a two-sided identity for $\otimes_{\text{endo}}$: for every context element C,

$$C \otimes_{\text{endo}} e = C \qquad \textit{and} \qquad e \otimes_{\text{endo}} C = C.$$

Proof. Let $C = (w^\star, d, p)$ be an arbitrary context element.

Right identity: $C \otimes_{\text{endo}} e = C$**.** Applying Definition 5.3 with $C_A = C$ and $C_B = e = (-\infty, 0, 0)$:

$$\begin{aligned} {w^\star}' &= \max(w^\star,\ -\infty) = w^\star, \\ {d_{\text{total}}}' &= d + 0 = d, \\ d'_{\text{pre}} &= p \quad \text{(since } w^\star \geq -\infty\text{, the first case of Equation (5.3) applies).} \end{aligned}$$

Therefore $C \otimes_{\text{endo}} e = (w^\star, d, p) = C$. ✓

Left identity: $e \otimes_{\text{endo}} C = C$**.** Applying Definition 5.3 with $C_A = e = (-\infty, 0, 0)$ and $C_B = C$:

$$\begin{aligned} {w^\star}' &= \max(-\infty,\ w^\star) = w^\star, \\ {d_{\text{total}}}' &= 0 + d = d. \end{aligned}$$

For d'_{pre}, we consider two subcases. If $w^\star = -\infty$, then ${w^\star}_A = {w^\star}_B = -\infty$, so ${w^\star}_A \geq {w^\star}_B$ and $d'_{\text{pre}} = d_{\text{pre},A} = 0 = p$ (since $p = 0$ when $w^\star = -\infty$ by Definition 5.1). If $w^\star > -\infty$, then ${w^\star}_B = w^\star > -\infty = {w^\star}_A$, so the second case of Equation (5.3) applies:

$$d'_{\text{pre}} = d_{\text{total}\,A} + d_{\text{pre},B} = 0 + p = p.$$

In either subcase, $d'_{\text{pre}} = p$. Therefore $e \otimes_{\text{endo}} C = (w^\star, d, p) = C$. ✓ □

Theorem 5.10 (Context monoid)**.** *The set of context elements equipped with endogenous composition and identity element e forms a monoid:*

$$\left(\mathcal{C},\ \otimes_{\text{endo}},\ e\right) \quad \textit{is a monoid.}$$

That is, $\otimes_{\text{endo}}$ is associative (Proposition 5.7), and $e = (-\infty, 0, 0)$ is a two-sided identity (Proposition 5.9).

5.5 Non-Commutativity

Remark 5.11 (Non-commutativity of $\otimes_{\text{endo}}$). The operator $\otimes_{\text{endo}}$ is **not** commutative. We have already seen a concrete counterexample: in Examples 5.5 and 5.6, the same two elements composed in opposite orders give different results. Specifically, with $A = (3, 5, 2)$ and $B = (7, 4, 1)$:

$$A \otimes_{\text{endo}} B = (7, 9, 6), \qquad B \otimes_{\text{endo}} A = (7, 9, 1).$$

These differ in their d_{pre} components: $6 \neq 1$.

The reason is structural. When ${w^\star}_B > {w^\star}_A$ and B is to the right of A, *all* of A's development capacity ($d_{\text{total}\,A} = 5$) gets promoted into the pre-pivot region, yielding $d_{\text{pre}} = 5 + 1 = 6$. But when B is to the *left* of A (i.e., we compute $B \otimes_{\text{endo}} A$), the pivot is already in the left block, so only B's own pre-pivot count matters: $d_{\text{pre}} = d_{\text{pre},B} = 1$.

In other words, temporal ordering matters. The left block's entire development capacity is promoted when the pivot shifts rightward, but no symmetric promotion occurs when the pivot remains in the left block. This asymmetry is an inherent feature of the endogenous context structure—it reflects the physical fact that events before the pivot and events after the pivot play fundamentally different roles.

5.6 Systems Implications

The algebraic properties established in this chapter have direct consequences for the computational implementation.

Remark 5.12 (Parallel tree reduction). Because $\otimes_{\text{endo}}$ is strictly associative (Proposition 5.7), the context reduction over a sequence of n events can be parallelised as a balanced binary tree reduction with $O(\log n)$ depth. At each level of the tree, adjacent pairs of context elements are composed independently, halving the number of elements. After $\lceil \log_2 n \rceil$ levels, a single root element remains.

The implementation in `src/holographic_tree.py`[1] realises this parallel structure. Each call to `append()` triggers $O(\log n)$ compositions in the worst case, maintaining the invariant that the forest encodes the full context of all events seen so far. The root summary can be queried at any time via `get_root_summary()`.

Remark 5.13 (Streaming composition). Associativity also enables a simple streaming protocol: process events one at a time, composing each new singleton context element into a running accumulator via a left fold. The accumulator at any point equals the context element for the entire event history observed so far.

The code path in `src/tropical_semiring.py` implements this strategy directly. The function `build_tropical_context()` performs a left fold over the event sequence, while the holographic tree in `src/holographic_tree.py` implements the parallel tree variant. Both produce identical results for any input sequence—a fact verified empirically by `test_05_holographic_exactness.py`. The left fold has $O(n)$ sequential depth but requires only $O(1)$ working memory (a single accumulator); the tree has $O(\log n)$ depth but requires $O(\log n)$ memory for the forest. The choice between them is a classical space–parallelism trade-off, and associativity guarantees that the choice cannot affect correctness.

[1] Specifically the `HolographicContextTree` class, which maintains a binomial carry forest of composed context elements.

5.7 Exercises

Exercise 5.1. Let $A = (10, 3, 1)$ and $B = (5, 6, 4)$. Compute $A \otimes_{\text{endo}} B$ and $B \otimes_{\text{endo}} A$. Verify that the two results differ, and explain which component differs and why.

Exercise 5.2. Let $A = (5, 2, 1)$, $B = (5, 3, 2)$, and $C = (5, 4, 3)$, where all three pivot keys are tied at $w^\star = 5$. The tie-breaking rule in Definition 5.3 is that the left block wins when ${w^\star}_A \geq {w^\star}_B$ (i.e., the first case of Equation (5.3) applies on ties).

(a) Compute $(A \otimes_{\text{endo}} B) \otimes_{\text{endo}} C$. What is d_{pre} in each intermediate and final result?

(b) Compute $A \otimes_{\text{endo}} (B \otimes_{\text{endo}} C)$. Verify that the final result matches part (a).

(c) Explain in words why tie-breaking in favour of the left block is essential for associativity.

Exercise 5.3. Let C be any context element with $w^\star > -\infty$. Prove that for any $n \geq 1$,

$$C \otimes_{\text{endo}} \underbrace{e \otimes_{\text{endo}} e \otimes_{\text{endo}} \cdots \otimes_{\text{endo}} e}_{n \text{ copies}} = C.$$

Hint. Use the identity property (Proposition 5.9) and induction, or argue directly from the definition.

Exercise 5.4. Consider a sequence of events in which *every* event is non-focal (there are no focal-actor events at all). Let C be the context element obtained by composing the singleton elements for this sequence.

(a) What are $w^\star$, d_{total}, and d_{pre} for the resulting context element C?

(b) A context element is called *feasible* at threshold k if $d_{\text{pre}} \geq k$. Can the all-non-focal context element ever be feasible? Explain.

(c) What does this say about the necessity of focal events for the endogenous pivot mechanism?

Chapter 6

The Tropical Lift

The monoid developed in Chapter 5 compresses any contiguous block of events into a triple $(w^\star, d_{\text{total}}, d_{\text{pre}})$. This triple tracks a single pivot—the strongest focal event—and records how much development capacity sits before it. The triple is enough to test feasibility at a fixed threshold k after the fact, but it discards information: we learn the *best* pivot's pre-pivot count yet know nothing about the second-best, or about pivots that fall just short of feasibility.

This chapter lifts the monoid into a richer representation—a *tropical context*—that replaces the scalar $w^\star$ with a weight vector indexed by pre-pivot development count. The resulting structure is a faithful extension of the original monoid: it reproduces exactly the same $w^\star$ and d_{pre}, while simultaneously answering feasibility queries at every threshold from 0 to k in a single pass. The composition rule for weight vectors turns out to be a simple shift-and-max operation—the tropical semiring structure from which the chapter takes its name.[1]

6.1 From Monoid to Weight Vectors

Recall the context element $C = (w^\star, d_{\text{total}}, d_{\text{pre}})$ from Definition 5.1. The scalar $w^\star$ is the weight of the best pivot in the block, and d_{pre} tells us how many non-focal events precede it. If $d_{\text{pre}} \geq k$, the pivot is feasible. But what if $d_{\text{pre}} = k - 1$? The monoid offers no recourse: the best pivot fell one short, and we have no record of whether a slightly weaker pivot might have $d_{\text{pre}} \geq k$.

The motivation for the tropical lift is exactly this: we want to know the best pivot weight achievable with *exactly* j pre-pivot development events, for every j from 0 to k.

Definition 6.1 (Tropical context)**.** A **tropical context** is a pair

$$T \;=\; (W,\; d_{\text{total}}),$$

where

(i) $W \in (\mathbb{R} \cup \{-\infty\})^{k+1}$ is the **weight vector**. The entry $W[j]$ for $j = 0, 1, \ldots, k$ is the maximum weight among all focal events in the block whose pre-pivot development count equals exactly j. If no such focal event exists, $W[j] = -\infty$.

(ii) $d_{\text{total}} \in \mathbb{N}$ is the **total development capacity**: the count of non-focal events in the block, exactly as in Definition 5.1.

[1] For background on tropical semirings, see e.g. Maclagan and Sturmfels, *Introduction to Tropical Geometry*, AMS, 2015.

The weight vector is indexed from 0 (a pivot with no preceding non-focal events) up to k (a pivot with at least k preceding non-focal events, where counts beyond k are capped at slot k).

The connection to the original monoid is immediate.

Remark 6.2 (Recovering the monoid). Given a tropical context $T = (W, d_{\text{total}})$, the monoid's best pivot weight and its pre-pivot count are

$$w^\star = \max_{0 \le j \le k} W[j], \qquad d_{\text{pre}} = \min\Big(\arg\max_{0 \le j \le k} W[j],\ k\Big).$$

The tropical representation is strictly richer than the triple: it records the best pivot at *every* slot, not just the globally best one.

Feasibility also takes a clean form.

Remark 6.3 (Feasibility). A tropical context T is **feasible** at threshold k if and only if $W[k] > -\infty$. That is, there exists at least one focal event in the block with k or more non-focal events preceding it.

6.2 Lifting Single Events

Every individual event is lifted into the tropical context by the factory function `from_event`.

Definition 6.4 (Singleton tropical context)**.** Given an event e and a threshold k, the **singleton tropical context** is $T_e = (W_e, d_{\text{total}_e})$ defined as follows.

(i) If e is **focal** (a pivot candidate):

$$W_e[0] = w(e), \qquad W_e[j] = -\infty \ \text{ for } j = 1, \ldots, k, \qquad d_{\text{total}_e} = 0.$$

A focal event occupies slot 0: it is a pivot with zero development events before it.

(ii) If e is **non-focal** (a development event):

$$W_e[j] = -\infty \ \text{ for all } j = 0, \ldots, k, \qquad d_{\text{total}_e} = 1.$$

A non-focal event contributes one unit of development capacity but cannot serve as a pivot.

Example 6.5 (Focal singleton)**.** Let e be a focal event with weight 7.5 and let $k = 3$. Then

$$T_e = \big([7.5,\ -\infty,\ -\infty,\ -\infty],\ \ 0\big).$$

The pivot sits in slot 0 with weight 7.5; no development capacity exists.

Example 6.6 (Non-focal singleton)**.** Let e be a non-focal event and let $k = 3$. Then

$$T_e = \big([-\infty,\ -\infty,\ -\infty,\ -\infty],\ \ 1\big).$$

All weight-vector entries are $-\infty$ (no pivot is available), and the block contributes one unit of development capacity.

6.3 Tropical Composition

This section presents the central algorithm. Given two tropical contexts representing adjacent blocks—the left block T_A followed in time by the right block T_B—we compute their composition.

Definition 6.7 (Tropical composition $\otimes$)**.** Let $T_A = (W_A, d_A)$ and $T_B = (W_B, d_B)$ be tropical contexts with the same threshold k. Their **tropical composition** is

$$T_A \otimes T_B \;=\; (W_{\text{result}},\; d_A + d_B),$$

where W_{result} is computed by the following shift-and-max rule:

(i) Initialise $W_{\text{result}} := W_A$ (copy all of A's entries).

(ii) For each $j' = 0, 1, \ldots, k$, let $j = \min(j' + d_A,\; k)$ and set

$$W_{\text{result}}[j] \;\leftarrow\; \max\big(W_{\text{result}}[j],\; W_B[j']\big). \tag{6.1}$$

The logic is as follows. A pivot from block A that already had j pre-pivot development events keeps its slot: step (i) copies all of A's entries into the result. A pivot from block B that had j' pre-pivot development events within B acquires all d_A of A's non-focal events as additional predecessors, so step (ii) writes it to result slot $\min(j' + d_A,\; k)$. When $j' + d_A > k$, the shifted slot is capped at k (both slots $j' + d_A$ and k indicate "at least k pre-pivot events"), and the max accumulates multiple B-pivots that map to the same capped slot.

This is precisely the operational meaning of the d_{pre} promotion from Equation (5.3) in Chapter 5: when a pivot resides in block B, all of A's development capacity becomes pre-pivot.

Remark 6.8 (Capping at k). The iteration in Equation (6.1) handles capping explicitly: a pivot from B at slot j' with $j' + d_A > k$ is written to result slot $\min(j' + d_A,\; k) = k$, accumulating via max with any value already at slot k. This ensures that pivots with "at least k" pre-pivot events are correctly aggregated.

Example 6.9 (Tropical composition)**.** Let $k = 3$, and consider the two tropical contexts

$$T_A = \big([5.0,\; 3.0,\; -\infty,\; -\infty],\;\; d_A = 2\big),$$
$$T_B = \big([7.0,\; -\infty,\; 4.0,\; -\infty],\;\; d_B = 3\big).$$

We compute $T_A \otimes T_B$ using the iteration form. Initialise $W_{\text{result}} = W_A = [5.0,\; 3.0,\; -\infty,\; -\infty]$.

$j' = 0$: $j = \min(0 + 2,\; 3) = 2$. $W_{\text{result}}[2] \leftarrow \max(-\infty,\; W_B[0]) = \max(-\infty,\; 7.0) = 7.0$.

$j' = 1$: $j = \min(1 + 2,\; 3) = 3$. $W_{\text{result}}[3] \leftarrow \max(-\infty,\; W_B[1]) = \max(-\infty,\; -\infty) = -\infty$.

$j' = 2$: $j = \min(2 + 2,\; 3) = 3$. $W_{\text{result}}[3] \leftarrow \max(-\infty,\; W_B[2]) = \max(-\infty,\; 4.0) = 4.0$.

$j' = 3$: $j = \min(3 + 2,\; 3) = 3$. $W_{\text{result}}[3] \leftarrow \max(4.0,\; W_B[3]) = \max(4.0,\; -\infty) = 4.0$.

The result is

$$T_A \otimes T_B = \big([5.0,\; 3.0,\; 7.0,\; 4.0],\;\; d = 5\big).$$

Interpretation. The best pivot with zero pre-pivot development events has weight 5.0 (from A, slot 0). With exactly two pre-pivot events, the best weight is 7.0 (from B, shifted by $d_A = 2$). Slot 3 has weight 4.0: this comes from $W_B[2]$, which maps to slot $\min(2 + 2,\; 3) = 3$ via capping. The context is **feasible** at $k = 3$ with best feasible pivot weight 4.0.

Algorithm 4 Build tropical context from an event sequence.

```
procedure BUILDTROPICALCONTEXT(events, k)
    T ← EMPTY(k)                                   ▷ W = [−∞, . . . , −∞], d_total = 0
    for each event e in temporal order do
        T ← T ⊗ FROMEVENT(e, k)
    end for
    return T
end procedure
```

Proposition 6.10 (Associativity of tropical composition)**.** *For any tropical contexts T_A, T_B, and T_C with common threshold k,*

$$(T_A \otimes T_B) \otimes T_C \;=\; T_A \otimes (T_B \otimes T_C).$$

Proof. The d_{total} component of both sides equals $d_A + d_B + d_C$ by associativity of addition. It remains to verify the weight vector.

A pivot originating in block $X \in \{A, B, C\}$ at internal slot j_X ends up in result slot $s(j_X)$ defined by

$$s_A(j_A) = j_A, \qquad s_B(j_B) = j_B + d_A, \qquad s_C(j_C) = j_C + d_A + d_B,$$

all capped at k. These slot assignments are determined by the block's position in the concatenated sequence and are independent of parenthesisation. At each result slot j, both parenthesisations compute max over the same set of contributing pivots from A, B, and C, so the result is identical.

In detail, consider the left-associated computation. Let $T_{AB} = T_A \otimes T_B$ with $d_{AB} = d_A + d_B$. Then in $T_{AB} \otimes T_C$, a pivot from C at internal slot j_C shifts to $\min(j_C + d_{AB},\, k) = \min(j_C + d_A + d_B,\, k)$. A pivot from A at slot j_A was placed at slot j_A in T_{AB} and remains at slot j_A in the final result. A pivot from B at slot j_B was placed at slot $\min(j_B + d_A,\, k)$ in T_{AB} and remains there.

The right-associated computation yields the same slot assignments: $T_{BC} = T_B \otimes T_C$ places a C-pivot at $\min(j_C + d_B, k)$ and a B-pivot at slot j_B. Then $T_A \otimes T_{BC}$ leaves A-pivots at j_A, shifts B-pivots to $\min(j_B + d_A, k)$, and shifts C-pivots to $\min(j_C + d_B + d_A, k)$.

Since both parenthesisations produce the same slot assignment for every pivot and take max at each slot, the weight vectors agree. □

6.4 Building Context from a Sequence

With the composition operator in hand, computing the tropical context for an entire event sequence is a left fold.

Algorithm 4 is the direct implementation of the left fold. The code path is `src/tropical_semiring.py`: `build_tropical_context()`, which iterates over events and composes each singleton into the accumulator via `compose_tropical()`. We now trace through a complete example to build intuition.

Example 6.11 (Full step-by-step construction)**.** Let $k = 3$ and consider the sequence of five events:

$$e_1 \text{ (non-focal)}, \quad e_2 \text{ (focal, } w = 4), \quad e_3 \text{ (non-focal)}, \quad e_4 \text{ (focal, } w = 7), \quad e_5 \text{ (non-focal)}.$$

Initialisation. $T = ([-\infty,\ -\infty,\ -\infty,\ -\infty],\ \ d = 0)$.

Step 1: append e_1 (non-focal). $T_{e_1} = ([-\infty,\ -\infty,\ -\infty,\ -\infty],\ \ d = 1)$. Composing $T \otimes T_{e_1}$: the left block has $d = 0$, so B's entries shift by 0. Both weight vectors are all $-\infty$.

$$T = ([-\infty,\ -\infty,\ -\infty,\ -\infty],\ \ d = 1).$$

Step 2: append e_2 (focal, $w = 4$). $T_{e_2} = ([4,\ -\infty,\ -\infty,\ -\infty],\ \ d = 0)$. Composing with $d_A = 1$:

$$\begin{aligned}
&j = 0\colon\ j < 1, \quad W[0] = -\infty.\\
&j = 1\colon\ j \geq 1, \quad W[1] = \max(-\infty,\ T_{e_2}.W[0]) = \max(-\infty, 4) = 4.\\
&j = 2\colon\ j \geq 1, \quad W[2] = \max(-\infty,\ T_{e_2}.W[1]) = \max(-\infty, -\infty) = -\infty.\\
&j = 3\colon\ \text{similarly } -\infty.
\end{aligned}$$

$$T = ([-\infty,\ 4,\ -\infty,\ -\infty],\ \ d = 1).$$

Event e_2 (weight 4) lands in slot 1 because one non-focal event (e_1) precedes it.

Step 3: append e_3 (non-focal). $T_{e_3} = ([-\infty,\ -\infty,\ -\infty,\ -\infty],\ \ d = 1)$. Composing with $d_A = 1$: W_A entries stay, and W_B is all $-\infty$, so no new pivots appear. Only d_{total} increases.

$$T = ([-\infty,\ 4,\ -\infty,\ -\infty],\ \ d = 2).$$

Step 4: append e_4 (focal, $w = 7$). $T_{e_4} = ([7,\ -\infty,\ -\infty,\ -\infty],\ \ d = 0)$. Composing with $d_A = 2$:

$$\begin{aligned}
&j = 0\colon\ j < 2, \quad W[0] = -\infty.\\
&j = 1\colon\ j < 2, \quad W[1] = 4.\\
&j = 2\colon\ j \geq 2, \quad W[2] = \max(-\infty,\ T_{e_4}.W[0]) = \max(-\infty, 7) = 7.\\
&j = 3\colon\ j \geq 2, \quad W[3] = \max(-\infty,\ T_{e_4}.W[1]) = \max(-\infty, -\infty) = -\infty.
\end{aligned}$$

$$T = ([-\infty,\ 4,\ 7,\ -\infty],\ \ d = 2).$$

Event e_4 (weight 7) lands in slot 2: two non-focal events (e_1, e_3) precede it in the sequence.

Step 5: append e_5 (non-focal). $T_{e_5} = ([-\infty,\ -\infty,\ -\infty,\ -\infty],\ \ d = 1)$. Composing with $d_A = 2$: no new pivots; d_{total} increments.

$$T = ([-\infty,\ 4,\ 7,\ -\infty],\ \ d = 3).$$

Final result. $T = ([-\infty,\ 4,\ 7,\ -\infty],\ \ d = 3)$. Since $W[3] = -\infty$, the context is **not feasible** at $k = 3$. The best weight overall is $\max(W) = 7$, achieved at slot 2. This means the best pivot (e_4, weight 7) has exactly 2 pre-pivot development events.

If the threshold were $k = 2$, the context would be feasible with $W[2] = 7$.

Remark 6.12 (Brute-force verification). Example 6.11 can be verified by direct enumeration. The focal events and their non-focal predecessor counts are:

- e_2 (weight 4): preceded by e_1 (non-focal), so $d_{\text{pre}} = 1$.
- e_4 (weight 7): preceded by e_1, e_3 (non-focal), so $d_{\text{pre}} = 2$.

Therefore $W[1] = 4$ and $W[2] = 7$, with all other slots $-\infty$. This matches the left-fold result exactly.

6.5 Best Feasible Pivot Search

With the weight vector in hand, the feasibility query reduces to an index lookup.

Definition 6.13 (Best feasible pivot)**.** Given a tropical context $T = (W, d_{\text{total}})$ and threshold k, the **best feasible pivot weight** is

$$w^\star{}_{\text{feas}} = W[k].$$

If $W[k] = -\infty$, no feasible pivot exists. If $W[k] > -\infty$, the best feasible pivot has weight $W[k]$ and at least k non-focal predecessors.

To recover the *identity* of the best feasible pivot (not just its weight), the algorithm must trace back through the composition to find which event contributed $W[k]$. In the streaming setting, this requires maintaining the event reference alongside the weight. In the holographic tree setting (Remark 5.12 in Chapter 5), the tree's structure enables efficient pivot-block lookup.

Remark 6.14 (Top-m search). For applications requiring the top m feasible pivots rather than just the best, the function `focal_pivots_with_prefix` enumerates all focal events together with their prefix development counts. Filtering to those with prefix count $\geq k$ and sorting by weight yields the top-m list. This brute-force enumeration runs in $O(n \log n)$ time (dominated by sorting) and is used for validation purposes.

6.6 Subsumption of the Original Monoid

The tropical context is designed as a strict generalisation of the original monoid. We now state the precise subsumption relationship and report its empirical validation.

Proposition 6.15 (Tropical subsumption)**.** *Let $T = (W, d_{\text{total}})$ be the tropical context obtained by left-fold composition over an event sequence, and let $C = (w^\star, d_{\text{total}}, d_{\text{pre}})$ be the original monoid context computed over the same sequence. Then*

$$\max_{0 \leq j \leq k} W[j] = w^\star, \tag{6.2}$$

$$\arg\max_{0 \leq j \leq k} W[j] = \min(d_{\text{pre}},\ k), \tag{6.3}$$

where arg max *returns the smallest index achieving the maximum in the case of ties.*

Proof sketch. Both the monoid and the tropical context process the same event sequence by left fold. The monoid tracks only the globally strongest pivot (breaking ties in favour of the leftmost), while the tropical context tracks the strongest pivot *at each slot*. The globally strongest pivot occupies exactly one slot, say slot j^*, so $\max(W) = W[j^*] = w^\star$.

For Equation (6.3), the slot j^* records the number of non-focal predecessors of the global pivot, capped at k. The monoid's d_{pre} is the uncapped count, so $j^* = \min(d_{\text{pre}}, k)$. □

Remark 6.16 (Empirical validation). The test suite `test_03_monoid_subsumption.py` validates Proposition 6.15 across 200 random seeds per configuration, spanning a range of sequence lengths, focal ratios, and thresholds k. Across all configurations, zero violations are observed with a numerical tolerance of 10^{-12}. The tropical context faithfully reproduces the original monoid's outputs.

The tropical context is strictly richer than the monoid: it records feasibility information at every slot simultaneously. This per-slot resolution is what enables the contract-guarded compression developed in Chapter 7.

6.7 Complexity Analysis

Proposition 6.17 (Complexity of tropical operations)**.** *The following complexity bounds hold:*

(i) ***Single composition.*** *Computing $T_A \otimes T_B$ requires $O(k)$ time: one pass over the $k+1$ result slots.*

(ii) ***Left-fold construction.*** *Building the tropical context for a sequence of n events via Algorithm 4 requires $O(n \cdot k)$ time: n compositions, each $O(k)$.*

(iii) ***Holographic tree.*** *The tree variant maintains $O(\log n)$ composition depth, achieving $O(k \log n)$ amortised cost per event insertion and $O(n \cdot k)$ total build cost.*

Remark 6.18 (Practical cost). In the narrative application, the threshold k is typically small ($1 \leq k \leq 5$). For such values the $O(k)$ factor is a small constant, and the left-fold construction is effectively $O(n)$. The holographic tree's $O(\log n)$ query depth becomes relevant for interactive or streaming applications where individual updates must be fast relative to the total sequence length.

6.8 Implementation Notes

Remark 6.19 (Sentinel value for $-\infty$). The implementation in `src/tropical_semiring.py` represents the weight vector W as a `numpy` array of floating-point numbers. The value $-\infty$ is encoded as Python's `float("-inf")`. The composition function `compose_tropical` implements the shift-and-max rule from Definition 6.7: it copies W_A into the result, then iterates over entries of W_B, shifting each by d_A and capping at k. This is equivalent to Equation (6.1).

Remark 6.20 (Factory methods). The `TropicalContext` class provides three factory methods:

- `empty(k)`: returns the identity element $([-\infty, \ldots, -\infty],\ 0)$.
- `from_event(e, k)`: implements Definition 6.4—for focal events, $W[0] = w(e)$ and $d_{\text{total}} = 0$; for non-focal events, all entries of W are $-\infty$ and $d_{\text{total}} = 1$.
- `compose(other)`: delegates to `compose_tropical`.

The left fold in `build_tropical_context` iterates over the event sequence, composing each singleton into the accumulator. The ground-truth validator `brute_force_tropical_context` enumerates all focal events, counts their non-focal predecessors directly, and populates the weight vector by brute force—providing an independent reference for testing.

6.9 Exercises

Exercise 6.1. Let $k = 2$ and consider the two tropical contexts

$$T_A = \big([6.0,\ -\infty,\ 2.0],\ \ d_A = 1\big), \qquad T_B = \big([8.0,\ 3.0,\ -\infty],\ \ d_B = 2\big).$$

(a) Compute $T_A \otimes T_B$ using the shift-and-max rule (Equation (6.1)).

(b) Is the result feasible at $k = 2$?

(c) What are $w^\star$ and d_{pre} for the result? Verify that they match what the original monoid would produce.

Exercise 6.2. Show that the empty tropical context $T_\varepsilon = ([-\infty, \ldots, -\infty],\ 0)$ is a two-sided identity for $\otimes$. That is, for any tropical context T,

$$T \otimes T_\varepsilon = T \qquad \text{and} \qquad T_\varepsilon \otimes T = T.$$

Hint. For the left-identity case, note that $d_A = 0$ in the shift-and-max rule.

Exercise 6.3. Let $k = 2$ and consider the event sequence:

$$e_1 \text{ (focal, } w = 5), \quad e_2 \text{ (non-focal)}, \quad e_3 \text{ (focal, } w = 3), \quad e_4 \text{ (non-focal)}.$$

Trace through the left-fold construction (Algorithm 4) step by step. State the tropical context after each event is appended. Verify the final result against brute-force enumeration of the focal events and their predecessor counts.

Exercise 6.4. Let all n events in a sequence be non-focal. What is the tropical context produced by Algorithm 4? Compare this to the monoid context from Exercise 5.4 in Chapter 5 and confirm that the subsumption relation (Proposition 6.15) holds.

Exercise 6.5. Let $k = 2$ and define

$$\begin{aligned} T_A &= \big([4.0,\ -\infty,\ -\infty],\ \ d_A = 1\big), \\ T_B &= \big([-\infty,\ -\infty,\ -\infty],\ \ d_B = 1\big), \\ T_C &= \big([9.0,\ -\infty,\ -\infty],\ \ d_C = 0\big). \end{aligned}$$

Compute $(T_A \otimes T_B) \otimes T_C$ and $T_A \otimes (T_B \otimes T_C)$ separately. Verify that the results are identical and that the final context is feasible at $k = 2$.

Chapter 7

The Absorbing Ideal and Compression Contracts

The preceding chapters have established two independent lines of reasoning. On the algebraic side, Chapter 5 gave us the endogenous context monoid $(\mathcal{C}, \otimes_{\text{endo}})$ with strict associativity and a logarithmic parallel reduction. On the structural side, Chapter 3 showed that prefix-deficient states are absorbing under greedy semantics—once a sequence has committed to a deficient prefix, no continuation can rescue it.

This chapter unifies the two lines. We extend the context element with a *commitment flag*, define a richer composition operation $\otimes_{\text{commit}}$ that respects commitment, and prove that the set of absorbing elements forms a *left ideal* in the resulting monoid. We then turn to the practical question: which operations can push an element across the absorbing boundary? The answer, confirmed both by theory and by experiment, is that *compression is the unique closure-breaking operation*. We conclude by formulating a *no-absorption contract* that every compression map must satisfy to preserve algebraic closure.

7.1 Extended Context Elements

In Chapter 5 a context element was a triple $(w^\star, d_{\text{total}}, d_{\text{pre}})$. To model systems that have irrevocably selected a pivot, we adjoin a single bit.

Definition 7.1 (Extended context element)**.** An *extended context element* is a quadruple

$$\bar{C} \;=\; (w^\star,\; d_{\text{total}},\; d_{\text{pre}},\; \kappa)$$

where $w^\star \in \mathbb{R}$ is the weight of the focal event (running-max pivot weight), $d_{\text{total}} \in \mathbb{N}$ is the total development count (number of non-focal events in the block), $d_{\text{pre}} \in \mathbb{N}$ is the pre-pivot development count (non-focal events that precede the current pivot), and $\kappa \in \{0, 1\}$ is the *commitment flag*:

- $\kappa = 0$: **uncommitted**. The pivot identity is provisional; a future suffix may contain an event of higher weight that replaces the current pivot.
- $\kappa = 1$: **committed**. The pivot identity is locked. No event in any future suffix can replace it, regardless of weight.

We write $\bar{\mathcal{C}}$ for the set of all extended context elements.

Remark 7.2 (Operational interpretation of commitment). The commitment flag models any mechanism by which a system makes an irrevocable decision about pivot identity. Three common sources of commitment arise in practice:

(i) *Streaming assignment.* A streaming processor that emits labels on the fly has, at each step, assigned roles (pre-pivot development, pivot, post-pivot material) based on the current running-max pivot. Once a label has been emitted, the assignment is irrevocable: setting $\kappa = 1$ records this fact.

(ii) *Compressed context.* A solver that compresses its context window typically discards events that were "redundant" relative to a particular pivot. The compressed representation is thereafter committed to that pivot, because the discarded events are no longer available to support a different one.

(iii) *Checkpoint publication.* Any system that publishes an intermediate result—a partial narrative, a tentative ranking—has committed to the pivot underlying that result. Rolling back the pivot would invalidate the published output.

In each case, the system has passed a point of no return. The commitment flag is the algebraic encoding of that irreversibility.

7.2 Committed Composition

With extended context elements in hand, we define the composition operation that respects commitment.

Definition 7.3 (Committed composition $\otimes_{\text{commit}}$)**.** Given extended context elements

$$\bar{C}_A = (w^\star{}_A,\ d_A,\ d_{\text{pre},A},\ \kappa_A), \qquad \bar{C}_B = (w^\star{}_B,\ d_B,\ d_{\text{pre},B},\ \kappa_B),$$

their *committed composition* $\bar{C}_A \otimes_{\text{commit}} \bar{C}_B = (w^\star,\ d_{\text{total}},\ d_{\text{pre}},\ \kappa)$ is defined by three cases.

Case 1. $\kappa_A = 1$ *(left block committed).*

$$w^\star = w^\star{}_A, \qquad d_{\text{total}} = d_A + d_B, \qquad d_{\text{pre}} = d_{\text{pre},A}, \qquad \kappa = 1.$$

The committed pivot in A is preserved unconditionally. Block B contributes to the total development count but cannot override the pivot, regardless of $w^\star{}_B$.

Case 2. $\kappa_A = 0$ and $w^\star{}_A \geq w^\star{}_B$ *(left uncommitted, left pivot dominates).*

$$w^\star = w^\star{}_A, \qquad d_{\text{total}} = d_A + d_B, \qquad d_{\text{pre}} = d_{\text{pre},A}, \qquad \kappa = \kappa_A = 0.$$

This coincides with the endogenous composition $\otimes_{\text{endo}}$: the higher-weight pivot stays in A.

Case 3. $\kappa_A = 0$ and $w^\star{}_B > w^\star{}_A$ *(left uncommitted, right pivot dominates).*

$$w^\star = w^\star{}_B, \qquad d_{\text{total}} = d_A + d_B, \qquad d_{\text{pre}} = d_A + d_{\text{pre},B}, \qquad \kappa = \kappa_B.$$

The pivot shifts to B. All of A's events become pre-pivot development material and are added to $d_{\text{pre},B}$. The commitment status of the result inherits from B.

Remark 7.4 (Comparison with $\otimes_{\text{endo}}$). The critical difference between $\otimes_{\text{commit}}$ and $\otimes_{\text{endo}}$ lies exclusively in Case 1. Under $\otimes_{\text{endo}}$, there is no Case 1: if $w^\star{}_B > w^\star{}_A$, the pivot always shifts to B. Under $\otimes_{\text{commit}}$, the commitment flag blocks this shift. The pivot in A is preserved even when B contains a strictly higher-weight event.

Cases 2 and 3 of $\otimes_{\text{commit}}$ are identical to the two cases of $\otimes_{\text{endo}}$. Committed composition therefore extends endogenous composition: when $\kappa_A = 0$, the two operations agree.

Example 7.5 (Committed composition locks a suboptimal pivot)**.** Let $\bar{C}_A = (3,\ 5,\ 1,\ 1)$ and $\bar{C}_B = (10,\ 4,\ 2,\ 0)$.

Under $\otimes_{\text{endo}}$*:* Since $w^\star{}_B = 10 > 3 = w^\star{}_A$, the pivot shifts to B. The result is

$$\bar{C}_A \otimes_{\text{endo}} \bar{C}_B = (10,\ 5+4,\ 5+2,\ 0) = (10,\ 9,\ 7,\ 0).$$

The stronger pivot in B has been selected, and all five of A's events now contribute to the pre-pivot development count.

Under $\otimes_{\text{commit}}$: Since $\kappa_A = 1$, Case 1 applies regardless of weights. The result is

$$\bar{C}_A \otimes_{\text{commit}} \bar{C}_B = (3,\ 5+4,\ 1,\ 1) = (3,\ 9,\ 1,\ 1).$$

The system remains locked to the weight-3 pivot in A, ignoring the weight-10 event in B. The pre-pivot count stays at $d_{\text{pre}} = 1$. If $k = 3$ (the grammar's prefix requirement), then $d_{\text{pre}} = 1 < 3$ and the committed result is absorbing—a situation that $\otimes_{\text{endo}}$ would have avoided by shifting to the stronger pivot.

7.3 The Absorbing Predicate

We now give the algebraic characterisation of states from which the grammar's prefix requirement can never be satisfied.

Definition 7.6 (Absorbing predicate)**.** Fix a grammar prefix requirement $k \in \mathbb{N}$. The *absorbing predicate* $\perp$ is the subset of $\bar{\mathcal{C}}$ defined by

$$\bar{C} \in \perp \quad \Longleftrightarrow \quad d_{\text{pre}}(\bar{C}) < k.$$

An element $\bar{C}$ satisfying $\bar{C} \in \perp$ is said to be *absorbing*: its pre-pivot development count is insufficient to meet the prefix requirement.

Remark 7.7 (Connection to the impossibility theorem). The absorbing predicate is the algebraic generalisation of the condition that caused greedy failure in Chapter 3. In that chapter, the relevant quantity was $j_{\text{dev}} < k$: the development index of the current pivot was too small for the grammar to be satisfiable. Here, $d_{\text{pre}} < k$ plays exactly the same role, lifted from the sequential setting into the algebraic framework.

The key insight is that j_{dev} was defined relative to a specific sequence position, whereas d_{pre} is a property of an abstract context element. This abstraction is what allows the absorbing condition to compose: we can reason about the absorption status of a composite block without unfolding it into its constituent events.

7.4 Absorbing Left Ideal

The main algebraic result of this chapter is that the absorbing predicate defines a left ideal in the committed monoid. Informally: once a system has committed to an absorbing state, no suffix can rescue it.

Proposition 7.8 (Absorbing left ideal)**.** *Let $k \in \mathbb{N}$ be the grammar prefix requirement. If $\bar{C}_A \in \bar{\mathcal{C}}$ satisfies $\kappa_A = 1$ and $\bar{C}_A \in \bot$ (i.e., $d_{\mathrm{pre},A} < k$), then for every $\bar{C}_D \in \bar{\mathcal{C}}$,*

$$\bar{C}_A \otimes_{\mathrm{commit}} \bar{C}_D \ \in \ \bot.$$

That is, the set $\{\, \bar{C} \in \bar{\mathcal{C}} : \kappa = 1 \text{ and } d_{\mathrm{pre}} < k \,\}$ is a left ideal of $(\bar{\mathcal{C}}, \otimes_{\mathrm{commit}})$.

Proof. Since $\kappa_A = 1$, Case 1 of Definition 7.3 applies unconditionally, regardless of the values ${w^\star}_D$, d_D, $d_{\mathrm{pre},D}$, and κ_D carried by the suffix. The committed composition rule yields:

$$\bar{C}_A \otimes_{\mathrm{commit}} \bar{C}_D \ = \ \big({w^\star}_A,\ \ d_A + d_D,\ \ d_{\mathrm{pre},A},\ \ 1\big).$$

We verify each component of the result against the absorbing predicate and the left ideal requirement.

Pre-pivot count is preserved. The pre-pivot development count of the result is $d_{\mathrm{pre},A}$. The suffix $\bar{C}_D$ does not contribute to d_{pre}: because the pivot is locked in A, no event in D can become the new pivot, and therefore no event in D can change the set of events that precede the pivot. The pre-pivot count is determined entirely by the committed left block.

The absorbing predicate is satisfied. By hypothesis, $d_{\mathrm{pre},A} < k$. Since the pre-pivot count of the result equals $d_{\mathrm{pre},A}$, the result also has $d_{\mathrm{pre}} < k$. Therefore $\bar{C}_A \otimes_{\mathrm{commit}} \bar{C}_D \in \bot$.

The commitment flag is preserved. The result has $\kappa = 1$. Combined with $d_{\mathrm{pre}} < k$, this shows that the result lies in the set $\{\, \bar{C} : \kappa = 1 \text{ and } d_{\mathrm{pre}} < k \,\}$, confirming that the set is closed under right-multiplication by arbitrary elements of $\bar{\mathcal{C}}$.

Since $\bar{C}_D$ was arbitrary, the set of committed absorbing elements is a left ideal of $(\bar{\mathcal{C}}, \otimes_{\mathrm{commit}})$. □

Remark 7.9 (Interpretation). Proposition 7.8 is the algebraic formalisation of an operationally devastating fact: *once committed to a bad state, you cannot recover*. No matter how rich, how well-structured, or how long the suffix is, it cannot repair the pre-pivot deficiency because the commitment flag prevents the pivot from moving. The absorbing elements form a "black hole" in the monoid: any element that falls in—and is committed—can never escape.

This should be contrasted with the sequential impossibility theorem of Chapter 3, which established the same conclusion for a specific greedy algorithm. Proposition 7.8 is stronger: it applies to *any* process that respects committed composition, regardless of the algorithm used to generate the suffix.

7.5 Escape Under Endogenous Semantics

The absorbing left ideal depends crucially on commitment. Without it, escape is possible—and understanding when escape occurs is essential for the validity mirage analysis of Chapter 9.

Remark 7.10 (Escape from absorption without commitment). Suppose $\bar{C}_A$ has $d_{\text{pre},A} < k$ (so $\bar{C}_A \in \perp$) but $\kappa_A = 0$ (uncommitted). Under $\otimes_{\text{endo}}$, there is no Case 1: the pivot can shift if a higher-weight event appears on the right. If the suffix $\bar{C}_D$ has ${w^\star}_D > {w^\star}_A$ and $d_{\text{pre},D} \geq k - d_A$, then Case 3 of the composition applies:

$$\bar{C}_A \otimes_{\text{endo}} \bar{C}_D = ({w^\star}_D,\ \ d_A + d_D,\ \ d_A + d_{\text{pre},D},\ \ \kappa_D).$$

The new pre-pivot count is $d_A + d_{\text{pre},D}$. If $d_A + d_{\text{pre},D} \geq k$, then $\bar{C}_A \otimes_{\text{endo}} \bar{C}_D \notin \perp$: the composite has escaped absorption.

Example 7.11 (Absorption escape under $\otimes_{\text{endo}}$)**.** Let $k = 3$ and consider

$$\bar{C}_A = (3,\ 2,\ 0,\ 0).$$

Since $d_{\text{pre},A} = 0 < 3 = k$, we have $\bar{C}_A \in \perp$. Now let

$$\bar{C}_D = (10,\ 5,\ 4,\ 0).$$

Under endogenous composition, ${w^\star}_D = 10 > 3 = {w^\star}_A$, so the pivot shifts to D:

$$\bar{C}_A \otimes_{\text{endo}} \bar{C}_D = (10,\ 2+5,\ 2+4,\ 0) = (10,\ 7,\ 6,\ 0).$$

The pre-pivot count is $6 \geq 3 = k$: the composite has escaped absorption.

Under committed composition with $\kappa_A = 1$, the same inputs yield

$$\bar{C}_A \otimes_{\text{commit}} \bar{C}_D = (3,\ 7,\ 0,\ 1).$$

The pre-pivot count remains $0 < 3$: still absorbed. Commitment prevents escape.

Remark 7.12 (Algebraic foundation of the validity mirage). The contrast between committed and uncommitted semantics is the algebraic foundation of the *validity mirage* developed in Chapter 9 [5].

An enumerative solver with beam width $M > 1$ explores multiple pivot candidates simultaneously. Each candidate corresponds to a different factorisation of the sequence into pre-pivot, pivot, and post-pivot material. The solver's search process effectively operates under $\otimes_{\text{endo}}$ semantics: it can "escape" absorption by substituting a different pivot—one that was not the running-max focal event of the original prefix, but that appeared in a different branch of the beam.

The resulting output may well be valid: every grammar constraint is satisfied, and every structural requirement is met. But the pivot identity has changed. The solution that the solver returns is not the solution that the original prefix was building toward. Semantics have drifted silently, and no constraint violation has been raised. This is the mirage: the output *looks* correct because validity is preserved, yet the underlying semantic intent has been abandoned.

7.6 The No-Absorption Compression Contract

The left ideal theorem (Proposition 7.8) tells us that committed absorption is permanent. The natural follow-up question is: which operations can *create* absorption where none existed before? The empirical answer, detailed in Section 7.7, is that compression is the unique culprit. This section formulates the contract that prevents it.

Definition 7.13 (No-absorption contract)**.** Fix a grammar prefix requirement $k \in \mathbb{N}$. A *compression map* $\mu\colon \mathcal{E} \to \mathcal{E}$ (where $\mathcal{E}$ denotes the space of event blocks) satisfies the *no-absorption contract* if for every event block B,

$$d_{\text{total}}\big(\mu(B)\big) \;\geq\; \min\big(d_{\text{total}}(B),\, k\big).$$

Remark 7.14 (Operational interpretation). The contract permits compression to remove redundant development capacity—events beyond what is needed for the prefix requirement—but forbids it from reducing the development count below the survival threshold k. Concretely:

- If the original block has $d_{\text{total}}(B) \geq k$, then the compressed block must also have $d_{\text{total}}(\mu(B)) \geq k$. Compression may discard the excess, but it must retain at least k development events.
- If the original block has $d_{\text{total}}(B) < k$, then the contract requires only $d_{\text{total}}(\mu(B)) \geq d_{\text{total}}(B)$: the compression must not make the situation worse, but it is not required to conjure development events that were never there.

Remark 7.15 (Why d_{total} and not d_{pre}). The contract is stated in terms of d_{total} rather than d_{pre} for a fundamental reason: the compression map does not know which event will ultimately serve as the pivot, nor which events will appear in the subsequent suffix.

At compression time, the pivot identity may still be provisional ($\kappa = 0$), and a future suffix may shift the pivot to a different event. The pre-pivot count d_{pre} is defined relative to a specific pivot; if the pivot changes, d_{pre} changes with it. By contrast, d_{total} counts *all* non-focal events in the block, regardless of their position relative to any particular pivot.

Preserving $d_{\text{total}} \geq k$ ensures that no matter where the pivot ultimately lands, there is sufficient development material available for the prefix requirement to be satisfied. A contract based on d_{pre} would protect only the current pivot assignment and would become vacuous—or actively harmful—if the pivot subsequently shifted.

From definition to algorithm. Definition 7.13 specifies the *minimal* contract: a guard on d_{total} that ensures enough development capacity survives compression to meet the prefix requirement k. Algorithm 5 below implements a *stronger* guard that additionally verifies suffix-feasibility—i.e., that the tropical weight at rank k is preserved after each candidate removal. The stronger guard is not required by the theory (the absorbing-ideal results hold with the minimal contract) but is advisable in practice, since it catches near-absorbing states that the minimal $d_{\text{total}} \geq k$ contract would pass.

Remark 7.16 (Necessary vs. sufficient guards). Definition 7.13 states a *necessary* condition for no absorption: if $d_{\text{total}}(\mu(B)) < \min(d_{\text{total}}(B),\, k)$, absorption is guaranteed. The algorithm's tropical $W[k]$ guard is a *sufficient* condition: it checks whether the best feasible pivot—one with at least k non-focal predecessors—survives compression. The $W[k]$ guard is strictly stronger than the d_{total} contract: one can satisfy $d_{\text{total}} \geq k$ while placing all k non-focal events *after* every focal event, which destroys the pivot's pre-pivot prefix. The $W[k]$ guard catches this case; the d_{total} contract does not. In practice, use the $W[k]$ guard when tropical contexts are available and fall back to the d_{total} contract when only event counts are known.

Algorithm 5 presents the contract-guarded compression procedure. The algorithm iterates over non-focal events in a seed-determined random order, greedily attempting to remove each one. Before removing an event, it rebuilds the tropical context (Chapter 6) for the reduced block and checks whether the tropical weight at rank k is preserved. If removal would reduce $\mathbf{W}[k]$ below its original value, the event is blocked: it is essential for maintaining the prefix requirement.

Algorithm 5 Contract-guarded compression

Require: Event block *events*, prefix requirement k, retention fraction *retention* $\in (0, 1]$, random seed *seed*
Ensure: Compressed event block satisfying the no-absorption contract

```
target ← ⌈retention × |events|⌉
removable ← non-focal events in events, shuffled by seed
removed ← 0
for each candidate event e in removable do
    if |events| − removed ≤ target then
        break                                          ▷ Reached retention target
    end if
    ctx_without ← BUILDTROPICALCONTEXT(events \ {e}, k)
    if ctx_without.W[k] ≥ original.W[k] then           ▷ Feasibility preserved
        Remove e from events;   removed ← removed + 1
    else
        Block e                                        ▷ Removal would violate contract
    end if
end for
return remaining events
```

The greedy removal order (controlled by the random seed) means that the algorithm is not guaranteed to achieve the *maximum* compression ratio consistent with the contract. However, it is guaranteed to *satisfy* the contract: every removal is individually checked, and no removal that would violate feasibility is permitted. The retention parameter provides a hard floor on the number of events retained, offering an additional safety margin beyond the algebraic contract. The overhead of the contract guard is bounded: each candidate removal requires one call to BUILDTROPICALCONTEXT at cost $O(n \cdot k)$, and at most $O(n)$ candidates are tested, giving $O(n^2 k)$ worst-case cost—acceptable for the moderate block sizes typical of streaming applications.

7.7 Empirical Validation

The algebraic theory of the preceding sections makes precise predictions. We now report the experimental results that validate those predictions and, crucially, identify the one operation that breaks them.

7.7.1 Algebraic Core (Experiment 51)

Chapter 5 established that $\otimes_{\text{endo}}$ is strictly associative. The same property extends to $\otimes_{\text{commit}}$: committed composition inherits associativity from the case analysis, because Case 1 is idempotent (committed blocks remain committed) and the remaining cases reduce to $\otimes_{\text{endo}}$.

Pairwise exactness. Across 240 randomly generated pairs of extended context elements, the pairwise composition $\bar{C}_A \otimes_{\text{commit}} \bar{C}_B$ was computed and compared against a reference implementation that evaluates the full event-level sequence. Result: **0 violations out of 240 cases.**

Associativity. Across 80 randomly generated triples $(\bar{C}_A, \bar{C}_B, \bar{C}_C)$, the two bracketings $(\bar{C}_A \otimes_{\text{commit}} \bar{C}_B) \otimes_{\text{commit}} \bar{C}_C$ and $\bar{C}_A \otimes_{\text{commit}} (\bar{C}_B \otimes_{\text{commit}} \bar{C}_C)$ were compared component-wise. Result: **0 violations out of 80 cases.**

These results validate the monoid structure computationally.

7.7.2 Absorbing-Ideal Closure (Experiment 56)

To test Proposition 7.8 empirically, we generated 96 test cases, each consisting of a committed absorbing element $\bar{C}_A$ (with $\kappa_A = 1$ and $d_{\text{pre},A} < k$) composed on the right with a randomly generated suffix $\bar{C}_D$. In every case, we checked whether the result $\bar{C}_A \otimes_{\text{commit}} \bar{C}_D$ satisfied $d_{\text{pre}} < k$.

Result: **0 violations out of 96 cases.** The left ideal property holds exactly across the full test suite.

7.7.3 Closure Diagnostics (Experiment 57)

Which elementary operations preserve the monoid's algebraic structure, and which break it? Experiment 57 tested each operation independently for closure violations. The results are summarised in Table 7.1.

Table 7.1: Closure violation rates by operation (Experiment 57). Each operation was tested on a suite of randomly generated extended context elements. A violation occurs when the operation produces an output that crosses the absorbing boundary: a non-absorbing input yields an absorbing output, or vice versa.

Operation	**Violation rate**
Composition ($\otimes_{\text{commit}}$)	0.000
Compression	0.133
Pivot update	0.000
Split-at-point	0.000
Absorbing escape under compression	0.033

The results are striking. Every operation except compression has a violation rate of exactly zero: composition, pivot update, and split-at-point all preserve the algebraic structure perfectly. Compression alone violates closure, at a rate of 13.3%. Furthermore, 3.3% of compressions cause an *absorbing escape*: a committed absorbing element is compressed into a non-absorbing one, or—more dangerously—a non-absorbing element is compressed into an absorbing one.

Conclusion: Compression is the *unique closure-breaking operation* in the context algebra. This is the key empirical result of the chapter. Every other elementary operation preserves the monoid structure, but naive compression can push elements across the absorbing boundary. This finding motivates the no-absorption contract of Definition 7.13: it is precisely the guard needed to tame the one operation that breaks closure.

7.7.4 Absorption Escape Rates

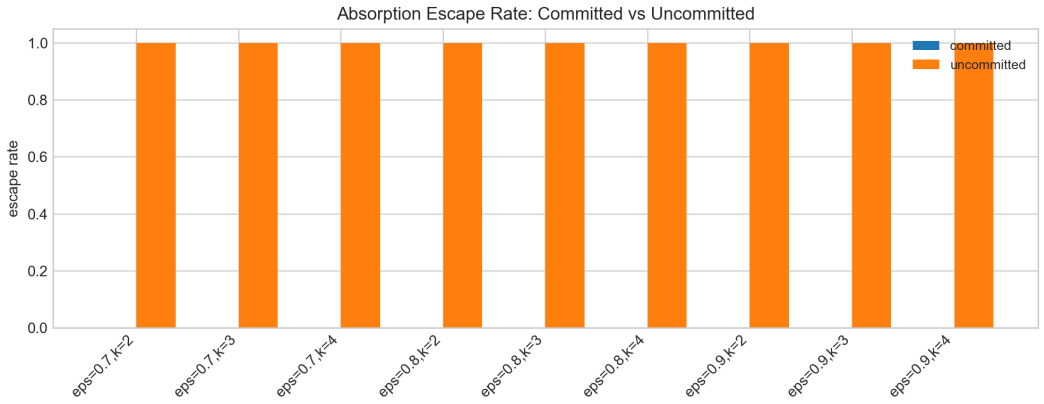


Figure 7.1: Absorption escape rates under committed versus uncommitted semantics across generator configurations. Under committed semantics ($\otimes_{\text{commit}}$), the escape rate is zero: the left ideal property holds. Under uncommitted semantics ($\otimes_{\text{endo}}$), escape is possible and occurs at measurable rates that depend on the weight distribution and block length.

Figure 7.1 displays the absorption escape rates for committed and uncommitted semantics across a range of generator configurations. The committed escape rate is uniformly zero, confirming Proposition 7.8. The uncommitted escape rate varies with the generator configuration but is consistently positive, illustrating the escape mechanism described in Section 7.5. The gap between the two curves is the algebraic signature of commitment-induced irreversibility.

7.8 Exercises

Exercise 7.1 (The absorbing ideal is not two-sided)**.** Show that the absorbing ideal of Proposition 7.8 is *not* a two-sided ideal. That is, construct extended context elements $\bar{C}_A$ and $\bar{C}_D$ such that $\bar{C}_A \in \perp$ with $\kappa_A = 1$, yet $\bar{C}_D \otimes_{\text{commit}} \bar{C}_A \notin \perp$.

Hint: Choose $\bar{C}_D$ with $\kappa_D = 0$ and a pivot of sufficiently high weight so that if $\bar{C}_D$ is on the left and uncommitted, the pivot stays in D. If $d_{\text{pre},D} \geq k$, the composite inherits the non-absorbing pre-pivot count from D.

Exercise 7.2 (Closure of absorbing elements under committed composition)**.** Prove that under committed semantics, the set of committed absorbing elements is closed under $\otimes_{\text{commit}}$ from the left. Specifically, show that if $\bar{C}_A \in \perp$ with $\kappa_A = 1$ and $\bar{C}_B \in \perp$ with $\kappa_B = 1$, then $\bar{C}_A \otimes_{\text{commit}} \bar{C}_B \in \perp$ with commitment flag $\kappa = 1$.

Exercise 7.3 (Maximum compression under the no-absorption contract)**.** Design a compression policy that satisfies the no-absorption contract (Definition 7.13) while achieving the maximum compression ratio. Specifically, for an event block B with $d_{\text{total}}(B) = 20$ non-focal events and grammar prefix requirement $k = 3$:

(a) What is the theoretical minimum number of non-focal events that must be retained?

(b) What is the maximum number of events that can be removed?

(c) Does the answer change if the block also contains focal events? Explain.

Exercise 7.4 (Why d_{total} and not d_{pre} in the contract)**.** Explain why the no-absorption contract (Definition 7.13) is stated in terms of d_{total} rather than d_{pre}. Construct a concrete scenario in which a compression map μ preserves $d_{\text{pre}}(\mu(B)) \geq k$ for the current pivot but produces a compressed block with $d_{\text{total}}(\mu(B)) < k$. Show that a subsequent pivot shift (under $\otimes_{\text{endo}}$ semantics) can then push the compressed block into the absorbing set, yielding $d_{\text{pre}} < k$ for the new pivot.

Chapter 8

Streaming Oscillation Traps

The preceding chapters analysed validity and absorption in an *offline* setting: the full event sequence is available before any labelling decision is made. In practice, however, events often arrive one at a time, and the system must emit labels incrementally. This chapter studies what goes wrong when the endogenous pivot problem is solved under streaming constraints [6].

The core phenomenon is an *oscillation trap*: the streaming policy commits to a pivot, then encounters a stronger focal event that invalidates the commitment. Because earlier labels are irrevocable, the policy is left in a state from which no continuation can produce a grammar-valid output. We prove that such traps are inevitable—the minimum inter-record gap in the pivot process is $O(1)$—and that they affect more than half of organically generated sequences. We then develop a *deferred commitment* policy that trades a small amount of latency for a dramatic reduction in trap rate, and characterise the quality–latency Pareto frontier.

We proceed as follows. Section 8.1 formalises the streaming extraction model and two concrete policies. Section 8.2 analyses the running-max pivot as a record process. Section 8.3 constructs adversarial sequences that trap the commit-now policy and proves the oscillation trap threshold. Section 8.4 reports the organic trap rate (Experiment 43). Section 8.5 validates the min_gap $< k$ predictor (Experiment 44). Section 8.6 demonstrates scale invariance (Experiment 45). Section 8.7 proves that the minimum inter-record gap is $O(1)$ under i.i.d. weights. Section 8.8 develops the deferred commitment policy and its Pareto analysis. Section 8.9 offers exercises.

8.1 Streaming Model Definitions

We begin with the formal streaming extraction model. The key distinction from the offline setting of Chapter 3 is *irrevocability*: once a label has been emitted, it cannot be changed.

Definition 8.1 (Streaming extraction policy)**.** A *streaming extraction policy*[1] is a function that processes an event sequence $e_1, e_2, \ldots, e_n$ in temporal order, maintaining a running state S_t at each step t. At each step the policy may assign an irrevocable label

$$\ell(e_t) \in \{\text{DEVELOPMENT}, \text{TURNING\_POINT}, \text{RESOLUTION}, \text{PENDING}\}$$

to the current event. Labels assigned in previous steps cannot be modified:

$$\text{if } \ell(e_s) \neq \text{PENDING at step } t, \quad \text{then } \ell(e_s) \text{ is fixed for all } t' > t.$$

[1] Implemented in `src/streaming.py`; the commit-now and buffered policies correspond to the `StreamingExtractor` class with `patience=0` and `patience=f`, respectively.

The policy's output is the final label assignment $(\ell(e_1), \ldots, \ell(e_n))$.

Definition 8.2 (Commit-now policy)**.** The *commit-now policy* is a streaming extraction policy that operates as follows. Let a^* denote the focal actor and let $\tau_t = \arg\max_{s \le t,\, a(e_s)=a^*} w(e_s)$ be the running-max pivot at step t. At each step t:

(i) If e_t is a focal event with $w(e_t) > w(\tau_{t-1})$ (or $t = 1$ and e_t is the first focal event), then e_t becomes the new running-max pivot: $\tau_t = e_t$. The policy immediately commits: it assigns $\ell(e_t) =$ TURNING_POINT and labels all pending non-focal events relative to e_t as the new pivot.

(ii) Labels assigned in previous steps are **irrevocable**. If τ_{t-1} was previously labelled TURNING_POINT and a new pivot $\tau_t \neq \tau_{t-1}$ is chosen, the old label cannot be retracted.

Definition 8.3 (Buffered policy with patience f)**.** The *buffered policy with patience* $f \in (0, 1)$ buffers events until step $\lceil f \cdot n \rceil$, then commits to the running-max pivot at that point. Formally:

(i) For $t \le \lceil f \cdot n \rceil$, all labels remain PENDING. The policy tracks the running-max pivot τ_t but does not emit any irrevocable labels.

(ii) At step $t = \lceil f \cdot n \rceil$, the policy commits to τ_t as the final pivot. All buffered events receive their labels relative to τ_t. Events arriving after step $\lceil f \cdot n \rceil$ receive committed labels immediately.

Events before the buffer point receive tentative labels that are finalised at commitment time; events after receive committed labels on arrival.

Definition 8.4 (Streaming absorbing trap)**.** A *streaming absorbing trap* is a state reached during streaming in which no continuation of the event sequence can produce a grammar-valid output under the commit-now policy. Formally, the policy is trapped at step t if:

(i) The running-max pivot has shifted from τ_{t-1} to $\tau_t \neq \tau_{t-1}$ (a *pivot shift*).

(ii) The number of non-focal events between τ_{t-1} and τ_t in the sequence is strictly less than k (the grammar's prefix requirement).

(iii) The labels assigned to events before τ_{t-1} under the old pivot assignment are irrevocable and cannot be reassigned as DEVELOPMENT for τ_t.

The trap is "sprung" when a pivot shift leaves too few development events between the old and new pivot to satisfy the prefix requirement.

8.2 Running-Max Pivot as Record Process

The running-max pivot τ_t updates only when a new maximum is encountered among focal events. This is precisely the definition of a *record* in the theory of order statistics.

The running-max pivot at step t is

$$\tau_t \;=\; \arg\max_{s \le t,\; a(e_s)=a^*} \; w(e_s).$$

The pivot updates at step t if and only if $w(e_t) > w(\tau_{t-1})$ and $a(e_t) = a^*$—that is, if e_t is a *record* among the focal events seen so far. Each such update is a *pivot shift*.

Under i.i.d. continuous weights, the theory of records [18, 20] gives sharp predictions about the frequency and timing of pivot shifts.

Proposition 8.5 (Expected number of records)**.** *If the weights of n focal events are drawn i.i.d. from a continuous distribution, the expected number of records (pivot shifts) is the n-th harmonic number:*

$$\mathbb{E}[\textit{number of records}] \;=\; H_n \;=\; \sum_{i=1}^{n} \frac{1}{i} \;\approx\; \ln n + \gamma,$$

where $\gamma \approx 0.5772$ is the Euler–Mascheroni constant.

Proof. The i-th observation is a record if and only if it is the maximum of the first i observations. For i.i.d. continuous random variables, each of the i observations is equally likely to be the maximum, so $\Pr(i\text{-th is a record}) = 1/i$. By linearity of expectation, the expected number of records in n observations is $\sum_{i=1}^{n} 1/i = H_n$. □

In practice, the event weights are not i.i.d.: the bursty generators used in our experiments impose a *front-loading* structure controlled by the parameter ε. Front-loading concentrates higher weights among earlier focal events, which affects both the number of pivot shifts and their timing.

Table 8.1: Pivot stability profile. Mean number of pivot shifts and median position of the last shift (as a fraction of the timeline), across front-loading parameter ε. Higher front-loading reduces the number of shifts and causes earlier stabilisation.

Front-loading ε	**Mean shifts**	**Median last-shift position**
0.05	4.64	0.481
0.10	4.16	0.436
0.20	3.72	0.382
0.40	3.31	0.312
0.60	3.09	0.265
0.80	2.95	0.224

Table 8.1 summarises the pivot stability profile. At low front-loading ($\varepsilon = 0.05$), the pivot shifts an average of 4.64 times, with the last shift occurring at a median position of 0.481 along the timeline—nearly halfway through. At high front-loading ($\varepsilon = 0.80$), the mean number of shifts drops to 2.95 and the pivot stabilises by roughly the first quarter.

Each pivot shift is a record in the weight sequence. The *cost* of a shift is proportional to the number of events labelled since the last shift—the "blast radius." Events labelled under the previous pivot assignment have irrevocable labels that may now conflict with the grammar's requirements under the new pivot. The larger the blast radius, the more damage a pivot shift inflicts on the streaming policy's output.

8.3 Adversarial Oscillation Traps

We now construct sequences that deterministically trap the commit-now policy. The construction makes the trapping mechanism explicit: record-setting focal events with strictly increasing weights are interleaved with non-focal events, and the oscillation period controls the spacing.

Definition 8.6 (Adversarial oscillation generator)**.** The *adversarial oscillation generator* with parameters (n, p, a^*) produces a sequence of n events as follows. Let p denote the *oscillation period*: the number of events between consecutive focal spikes. The generator places focal events at positions $1, p+1, 2p+1, \ldots$ with strictly increasing weights $w_1 < w_2 < w_3 < \cdots$. All other positions are

filled with non-focal events. The result is a sequence where every focal event is a record (a new running-max), and there are exactly $p - 1$ non-focal events between consecutive focal events.

The key quantity is the *effective pre-pivot capacity*: the maximum number of non-focal events available between consecutive record-setting focal events to serve as DEVELOPMENT for the new pivot.

Theorem 8.7 (Oscillation trap threshold)**.** *Consider a sequence produced by the adversarial oscillation generator (Definition 8.6) with oscillation period p. Under the commit-now policy (Definition 8.2) with grammar prefix requirement k, the policy is trapped if and only if*

$$\boxed{p_{\text{eff}} < k,}$$

where p_{eff} is the effective pre-pivot capacity: the maximum number of non-focal events between consecutive record-setting focal events that are available for DEVELOPMENT *assignment. For adversarial alternating-spike traces,*

$$p_{\text{eff}} = \max(1, \lfloor p/2 \rfloor).$$

Proof. We prove both directions.

(⇒) Trapping when $p_{\text{eff}} < k$. Suppose $p_{\text{eff}} < k$. Each time the running-max pivot shifts to a new focal event e_{new}, the commit-now policy fixes e_{new} as TURNING_POINT. Between the previous pivot e_{old} and e_{new}, there are at most p_{eff} non-focal events that could serve as DEVELOPMENT.

Since $p_{\text{eff}} < k$, the prefix requirement is not met for e_{new}: the grammar demands at least k DEVELOPMENT events before the TURNING_POINT, but only $p_{\text{eff}} < k$ are available in the interval between the two pivots.

Moreover, labels assigned to events before e_{old} under the old pivot assignment are irrevocable. These events were labelled relative to e_{old}—some as DEVELOPMENT for e_{old}, some as RESOLUTION—and these labels cannot be changed to DEVELOPMENT for e_{new}. Therefore the commit-now policy is trapped: the grammar requires k DEVELOPMENT events before the TURNING_POINT, but only $p_{\text{eff}} < k$ are available, and no continuation of the sequence can supply additional pre-pivot events because the TURNING_POINT is already fixed at e_{new}.

(⇐) No trap when $p_{\text{eff}} \geq k$. Suppose $p_{\text{eff}} \geq k$. Consider the final record—the last pivot shift in the sequence. Let e_{final} be the final pivot and e_{prev} the penultimate pivot. By assumption, there are at least $p_{\text{eff}} \geq k$ non-focal events between e_{prev} and e_{final}.

The commit-now policy can assign k of these non-focal events as DEVELOPMENT, satisfying the prefix requirement for e_{final}. Events after e_{final} can be labelled RESOLUTION. Since e_{final} is the last record, no further pivot shift will occur, so these labels remain valid. The grammar is satisfied. □

Table 8.2 reproduces the adversarial boundary for the commit-now policy across oscillation periods and prefix requirements. The boundary is sharp: validity transitions from 0% to 100% at the exact threshold predicted by Theorem 8.7.

8.4 Organic Prevalence (Experiment 43)

The adversarial construction of Section 8.3 demonstrates that traps are possible, but one might hope that they are pathological—artefacts of worst-case engineering that rarely occur in practice. Experiment 43 tests this hope against organically generated sequences.

Table 8.2: Commit-now validity (%) versus oscillation period for prefix requirements $k = 1, 2, 3$. The 0% $\to$ 100% boundary occurs at $p_{\text{eff}} = k$, exactly as predicted by Theorem 8.7.

Oscillation period p	$k = 1$	$k = 2$	$k = 3$
1	0	0	0
2	100	0	0
3	100	0	0
4	100	100	0
5	100	100	0
6	100	100	100
7	100	100	100
8	100	100	100

We generated 4,200 sequences using the bursty event generator across a grid of front-loading parameters $\varepsilon \in \{0.05, 0.10, 0.20, 0.40, 0.60, 0.80\}$ and prefix requirements $k \in \{1, 2, 3\}$, with 200 sequences per configuration. Each sequence was processed by the commit-now policy, and the output was checked for grammar validity.

Table 8.3: Commit-now trap rates on organic (bursty-generated) sequences. Trap rate is the fraction of sequences where the commit-now policy produces a grammar-invalid output, despite the existence of a valid offline solution.

	$k = 1$	$k = 2$	$k = 3$
Overall trap rate	38.9%	58.0%	67.9%
Peak trap rate	77% at $\varepsilon = 0.40$, $k = 3$		
Offline validity	83.5%–97.5%		

The headline result: **2,306 out of 4,200 organic sequences exhibit commit-now traps**—a trap rate of 54.9%. Table 8.3 breaks down the trap rate by prefix requirement k. Higher k means greater vulnerability: the grammar demands more pre-pivot development events, so a smaller inter-record gap is sufficient to spring the trap.

The peak trap rate of 77% occurs at $\varepsilon = 0.40$, $k = 3$. This is not a coincidence: moderate front-loading produces enough pivot shifts to create opportunities for trapping while concentrating records early where inter-record gaps tend to be small.

Meanwhile, finite (offline) validity remains in the 83.5%–97.5% range across the same configurations. The gap between offline feasibility and streaming feasibility is precisely the trap rate: these are sequences that *have* valid solutions, but the commit-now policy cannot find them because irrevocable early commitments block the path.

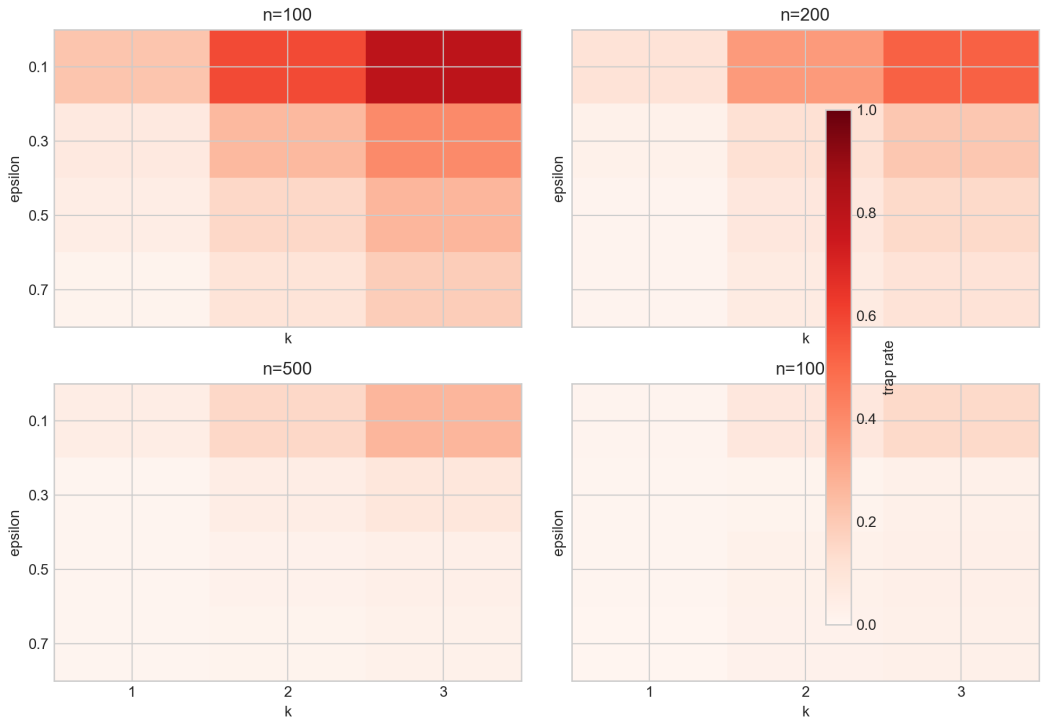


Figure 8.1: Organic trap rates by (ε, k). Commit-now traps affect more than half of organic sequences. The trap rate increases with k (rows) and peaks at moderate front-loading (columns). The gap between offline validity (high) and streaming validity (low) is the region where irrevocable commitment destroys feasibility.

Figure 8.1 presents the full heatmap.

8.5 Mechanism Verification (Experiment 44)

The oscillation trap threshold (Theorem 8.7) identifies $p_{\text{eff}} < k$ as the trapping condition for adversarial sequences. For organic sequences, the analogous predictor is the *minimum inter-record gap*: for each sequence, compute the minimum number of non-focal events between consecutive pivot shifts, and predict a trap whenever min_gap $< k$.

Definition 8.8 (Minimum inter-record gap)**.** Given a sequence with records (pivot shifts) at positions $r_1, r_2, \ldots, r_m$, the *inter-record gap* before the i-th record is the number of non-focal events between r_{i-1} and r_i. The *minimum inter-record gap* is

$$\text{min\_gap} \;=\; \min_{2 \le i \le m} \big(\text{non-focal events between } r_{i-1} \text{ and } r_i\big).$$

Experiment 44 evaluates the min_gap $< k$ predictor as a binary classifier for commit-now traps on the same 4,200 organic sequences from Experiment 43.

Table 8.4 presents the confusion matrix. The key results:

- **Accuracy:** 92.5% $\big((1{,}040 + 490)/(1{,}040 + 124 + 0 + 490)\big)$.
- **Recall:** 100% — zero false negatives. Every sequence that is actually trapped has min_gap $<$ k.

Table 8.4: Confusion matrix for the min_gap $< k$ trap predictor (Experiment 44). The predictor achieves 100% recall: every trapped sequence has min_gap $< k$.

	Predicted: Trap	**Predicted: No Trap**
Actual: Trap	1,040 (TP)	0 (FN)
Actual: No Trap	124 (FP)	490 (TN)

- **Precision:** 89.3% $\big(1{,}040/(1{,}040+124)\big)$.

The zero false negatives confirm that min_gap $< k$ is a **necessary condition** for trapping: if the minimum gap is at least k, the commit-now policy is guaranteed to succeed. The 124 false positives occur because having a small gap does not guarantee that the trap is sprung—the sequence may recover if later events provide sufficient development material after the final pivot shift.

Remark 8.9 (Necessary versus sufficient). The min_gap $< k$ condition is necessary but not sufficient for trapping. Sufficiency fails because a small gap at an intermediate record does not necessarily propagate to the final pivot assignment. If the *last* record has a gap $\geq k$, the final pivot has enough pre-pivot development events regardless of earlier gaps. The false positives are sequences where an intermediate record had min_gap $< k$ but the final pivot shift had a sufficient gap.

8.6 Scale Behaviour (Experiment 45)

One might conjecture that streaming traps are a small-sample artefact: perhaps as n grows, the increasing spacing between records gives the commit-now policy room to breathe. Experiment 45 tests this conjecture by sweeping n from 100 to 1,000.

The result is decisive: **trap rates remain in the** 52%–78% **band from** $n = 100$ **to** $n = 1{,}000$. There is no decay with scale. The mean minimum gap stays near 1.0 regardless of n.

Remark 8.10 (No escape through scale). The persistence of traps across scales is a direct consequence of the $O(1)$ minimum inter-record gap established in Proposition 8.11. Although the number of records grows as $\ln n$ and the *average* gap between records grows as $n/\ln n$, the *minimum* gap does not grow. The trap mechanism—which depends on the minimum gap, not the average—remains active at every practical scale.

8.7 Record-Gap Scaling (Proposition 2)

We now establish the theoretical foundation for the scale-invariance of streaming traps. The key result is that the minimum inter-record gap is $O(1)$ under i.i.d. continuous weights—it does not grow with n.

Proposition 8.11 (Minimum inter-record gap is $O(1)$)**.** *Let $X_1, X_2, \ldots, X_n$ be i.i.d. draws from a continuous distribution (no ties almost surely). Let $G_1, G_2, \ldots$ denote the inter-record gaps: G_i is the number of observations between the $(i-1)$-th and i-th records. Then:*

(i) For the first gap: $\Pr(G_1 = 1) = 1/2$.

(ii) The minimum gap satisfies $\min_i G_i = O(1)$ *almost surely. In particular,* $\Pr(\min_i G_i = 1) \to 1 - e^{-1} \approx 0.632$ *as* $n \to \infty$.

Proof. We proceed in two parts.

Part (i): The first gap. The first record is always X_1 (the first observation is trivially a record). The second record occurs at index j where $X_j > X_1$. The first inter-record gap is $G_1 = j - 1$.

The gap equals 1 if and only if $X_2 > X_1$—that is, the second observation is itself a new record. By exchangeability of continuous i.i.d. random variables, $\Pr(X_2 > X_1) = 1/2$. Therefore $\Pr(G_1 = 1) = 1/2$.

Part (ii): The minimum gap (Poisson approximation). Let $R_1 < R_2 < \cdots < R_M$ denote the positions of the M records among $X_1, \ldots, X_n$. We have $R_1 = 1$ always, and $M \approx \ln n$ in expectation. The gaps are $G_i = R_{i+1} - R_i$ for $i = 1, \ldots, M - 1$.

Key observation. If the i-th record occurs at position $R_i = j$, then the next observation X_{j+1} is itself a record (giving $G_i = 1$) if and only if X_{j+1} is the maximum of $X_1, \ldots, X_{j+1}$. By exchangeability,

$$\Pr(G_i = 1 \mid R_i = j) \;=\; \frac{1}{j+1}.$$

Poisson approximation. Let $N_n = \#\{i : G_i = 1\}$ count the number of unit gaps among the first $M - 1$ inter-record gaps. By a classical result on record spacings [18, Chapter 5], the number of unit gaps among the first n observations converges in distribution to a Poisson random variable with parameter $\lambda = 1$.

Therefore:

$$\Pr(\min_i G_i = 1) \;=\; \Pr(N_n \geq 1) \;=\; 1 - \Pr(N_n = 0) \;\to\; 1 - e^{-\lambda} \;=\; 1 - e^{-1} \;\approx\; 0.632.$$

The minimum gap is $O(1)$ almost surely: it equals 1 with probability tending to $1 - 1/e$, and equals a small constant otherwise. □

Remark 8.12 (Why traps persist across scales). Proposition 8.11 explains the empirical finding of Section 8.6: even though the number of records grows as $\ln n$, the gaps between them include arbitrarily small values (down to 1), and the minimum gap does not grow with n. The trap mechanism min_gap $< k$ remains active because the minimum gap is $O(1)$, not $O(n/\ln n)$. Increasing n adds more records with more opportunities for small gaps, rather than spacing existing records further apart.

Remark 8.13 (I.i.d. versus bursty weights). The proof of Proposition 8.11 uses i.i.d. continuous weights as a reference model. The bursty generators used empirically are *not* i.i.d.: they impose a front-loading structure in which higher weights are concentrated among earlier events (controlled by the parameter ε). The empirical trap rates are *higher* than the i.i.d. prediction because front-loading concentrates records early in the sequence, precisely where the inter-record gaps are most likely to be small.

Specifically, front-loading increases $\Pr(G_1 = 1)$ beyond the i.i.d. value of $1/2$, because early focal events are more likely to have high weights, making consecutive records in the first few positions more probable. The i.i.d. analysis therefore provides a *lower bound* on the trap rate for bursty sequences.

8.8 Deferred Commitment

The commit-now policy's fundamental weakness is that it commits irrevocably at each pivot shift, before the pivot has stabilised. Deferring commitment—waiting until a fraction f of the sequence has been observed before locking in the pivot—trades latency for validity. This section characterises the quality–latency Pareto frontier.

8.8.1 The Quality–Latency Pareto Curve

Experiment 48c evaluates the buffered policy (Definition 8.3) across patience values $f \in \{0, 0.10, 0.25, 0.50\}$ and compares against the finite (offline) solver as a ceiling. All evaluations use TP-consistent scoring (see Section 8.8.2).

Table 8.5: Quality–latency Pareto curve. Effective validity, score, and regret for the commit-now policy, deferred policies at three patience levels, and the offline ceiling. Deferred commitment at $f = 0.25$ recovers 80.1% validity versus commit-now's 39.4%.

Policy	Eff. Valid%	Eff. Score	Eff. Regret
Commit-now ($f = 0$)	39.4	6.68	9.90
Deferred $f = 0.10$	62.6	10.60	5.99
Deferred $f = 0.25$	80.1	14.17	2.42
Deferred $f = 0.50$	88.2	15.93	0.66
Finite (offline)	91.5	16.58	0.00

Table 8.5 presents the results. The headline: deferred commitment at $f = 0.25$ achieves 80.1% validity and a score of 14.17, compared with commit-now's 39.4% validity and 6.68 score. Deferred commitment dominates commit-now from $f = 0.05$ onward—every deferred policy along the Pareto frontier achieves both higher validity and higher score than the commit-now baseline.

At $f = 0.50$, the deferred policy reaches 88.2% validity, within 3.3 percentage points of the offline ceiling. The remaining gap represents sequences where the global-max pivot arrives after the halfway mark—a diminishing population as f increases.

8.8.2 TP-Consistent Scoring

Comparing streaming and offline policies requires care. The naive approach—scoring the streaming output against the offline gold standard—creates an arg max inconsistency. The streaming policy commits to a pivot τ_{stream}, while the offline solver selects a (potentially different) pivot τ_{offline}. If we score the streaming output using τ_{offline} as the reference, we are penalising the streaming policy for a choice it was forced to make under information constraints.

TP-consistent scoring resolves this by evaluating each policy against its own committed pivot. When scoring a deferred policy's output, we demote competing focal events—focal events other than the policy's committed pivot—so that the evaluation pivot matches the policy's pivot. This ensures that differences in score reflect structural quality (whether the grammar is satisfied, how many development events precede the pivot) rather than pivot identity disagreements.

Without TP-consistent scoring, a deferred policy that selects a slightly weaker pivot but satisfies the grammar perfectly would receive a lower score than an offline policy that selects the global-max pivot. The score difference would reflect the weight gap between pivots, not the structural quality of the output. TP-consistent scoring removes this confound.

8.8.3 Why $f = 0.25$

The choice $f = 0.25$ is not arbitrary. It is motivated by the empirical distribution of the global-max pivot's arrival time.

Remark 8.14 (Pivot arrival CDF). Let $F(t)$ denote the cumulative distribution function of the position (as a fraction of n) at which the global-max focal event first appears in the sequence. Across our

bursty generator configurations, $F(0.25) \approx 0.50$: the global-max pivot arrives before position 0.25 in roughly half of all instances.

A deferred policy with $f = 0.25$ therefore captures the correct pivot (the global max) in approximately 50% of sequences at commitment time, with no further pivot shifts possible after commitment. For the remaining 50%, the policy commits to a suboptimal pivot, but the grammar can often still be satisfied because the suboptimal pivot typically has sufficient pre-pivot development.

The $f = 0.25$ policy thus represents a favourable trade-off: it waits only through the first quarter of the event stream, captures the majority of pivot shifts, and recovers 80% of offline validity. Further increasing f yields diminishing returns: from $f = 0.25$ to $f = 0.50$, validity improves by only 8.1 percentage points at the cost of doubling the latency.

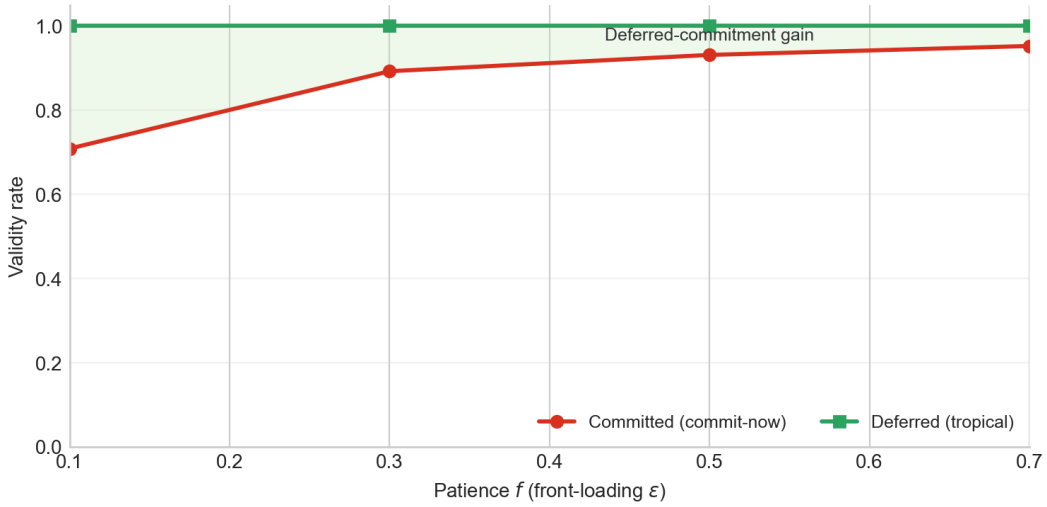


Figure 8.2: Quality–latency Pareto curve. Each point represents a streaming policy with patience f, plotted by effective validity (horizontal) and effective score (vertical). The commit-now policy ($f = 0$) occupies the lower-left corner; the offline ceiling ($f = 1$) the upper-right. Deferred policies trace a concave Pareto frontier, with $f = 0.25$ offering the best marginal return per unit of latency.

Figure 8.2 plots the full Pareto frontier.

8.9 Exercises

Exercise 8.1 (First inter-record gap)**.** Prove that for i.i.d. Uniform$[0, 1]$ weights, the probability that the first inter-record gap equals 1 is exactly $1/2$.

Hint. The first record is X_1. The gap equals 1 iff $X_2 > X_1$. Use exchangeability: for i.i.d. continuous random variables, $\Pr(X_2 > X_1) = 1/2$. This is a direct consequence of the proof of Proposition 8.11(i).

Exercise 8.2 (Front-loading and pivot arrival)**.** Analyse the effect of the front-loading parameter ε on the median pivot arrival fraction. Specifically:

(a) As ε increases (more weight concentrated among early events), what happens to the median fraction of the timeline at which the global-max pivot first appears?

(b) Why does early stabilisation of the pivot *not* eliminate streaming traps?

Answer. (a) The median pivot arrival fraction **decreases**: the global max occurs earlier because early focal events receive disproportionately high weights. The pivot stabilises sooner. (b) Early stabilisation means that the gap between the first and second records is likely to be very small—possibly 1—because the first few focal events are packed together with high weights. The pivot stabilises quickly, but the initial pivot shifts create small gaps that spring the trap before stabilisation occurs.

Exercise 8.3 (Step-by-step trap construction)**.** Construct a sequence of 8 events (with a focal actor a^* and prefix requirement $k = 2$) in which the commit-now policy is trapped but a deferred policy with $f = 0.5$ succeeds. Show the trap step by step:

(a) List the 8 events with their types (focal/non-focal) and weights.

(b) Trace the commit-now policy: show the running-max pivot, the labels emitted, and the step at which the trap is sprung.

(c) Trace the deferred policy with $f = 0.5$ (buffer until step 4): show that the pivot stabilises during the buffer period and the grammar is satisfied.

Hint. Use two focal events at positions 1 and 3 with weights $w_1 = 5$ and $w_3 = 10$, with a single non-focal event at position 2 between them. This gives a gap of $1 < k = 2$. Place non-focal events at positions 4–8 so the deferred policy has sufficient development material.

Exercise 8.4 (Why higher k means more traps)**.** Explain why increasing k makes streaming traps more likely. Your explanation should address:

(a) The relationship between k and the trapping condition min_gap $< k$.

(b) The $O(1)$ minimum gap result (Proposition 8.11) and why it implies that the trapping condition is easier to satisfy for larger k.

(c) The empirical trap rates from Table 8.3: 38.9% ($k = 1$), 58.0% ($k = 2$), 67.9% ($k = 3$).

Answer. (a) The trapping condition min_gap $< k$ becomes *easier* to satisfy as k grows: a gap of g that is harmless for $k \leq g$ becomes a trap for any $k > g$. (b) Since $\min_i G_i$ is $O(1)$—converging to 1 with probability 1 as $n \to \infty$ (Proposition 8.11)—the minimum gap is typically a small constant. For $k = 1$, a trap requires min_gap < 1, i.e., min_gap $= 0$ (consecutive focal records with no non-focal events between them), which is less common. For $k = 2$, min_gap $= 1$ suffices, which occurs with probability $\to 1$. For $k = 3$, min_gap ≤ 2 suffices, which is even more likely. (c) The empirical rates match this monotonic pattern: 38.9% $<$ 58.0% $<$ 67.9%.

Chapter 9

The Validity Mirage

The preceding chapters have established the algebra of context elements (Chapter 5), the tropical lift that tracks feasibility at every threshold (Chapter 6), and the absorbing ideal that makes committed failure permanent (Chapter 7). The theory predicts a specific pathology: a system can report perfect validity while silently substituting pivots, producing output that is well-formed yet semantically adrift. This chapter names that pathology, develops the diagnostic tools that expose it, and demonstrates it on both synthetic witnesses and real-world data.

The central empirical finding is stark. Raw validity—the fraction of instances for which the system produces *any* grammatically correct output—remains at or near 1.0 across all compression levels tested, from 90% retention down to 10%. But pivot preservation—the fraction of valid outputs that retain the *same* turning point as the uncompressed solve—collapses from near 1.0 to 0.354 over the same range. The system appears to be working perfectly. It is not. We call this gap the *validity mirage*.

9.1 Defining the Validity Mirage

Validity metrics answer a binary question: *does the output conform to the grammar?* Semantic fidelity answers a different question: *does the output preserve the intended meaning?* When raw feasibility stays high while semantic fidelity degrades, the metric conceals the failure. This is the mirage.

Definition 9.1 (Validity mirage)**.** Let S be an event sequence, τ_{full} the pivot selected by the full-sequence solve, and τ_{comp} the pivot selected by the compressed (or streamed) solve. A **validity mirage** exists for instance S when all four of the following conditions hold:

(i) The full-sequence solve is feasible with pivot τ_{full}.

(ii) The compressed solve is feasible with pivot τ_{comp}.

(iii) $\tau_{\text{comp}} \neq \tau_{\text{full}}$ *(pivot substitution has occurred)*.

(iv) No external signal—no error, no warning, no degraded-evidence flag—indicates that the substitution has taken place.

Condition (iv) is what makes the mirage dangerous. If the system raised a warning whenever it substituted a pivot, an operator could investigate. In the systems we study, no such warning exists. The compressed solve reports `valid = True`, the quality score is computed against the substitute pivot, and the output is delivered downstream without any indication that the semantic anchor has changed.

Remark 9.2 (Silent versus detectable mirages). We call the mirage **silent** when condition (iv) holds strictly: the system's own reporting channel provides no way to distinguish the mirage from a genuine success. All four diagnostic metrics developed in Section 9.2 are *external* to the system—they require access to the full-sequence solve for comparison. Without that external reference, the mirage is invisible.

9.2 The Three Semantic Diagnostics

To expose the mirage, we introduce three metrics that compare the compressed solve against the full-sequence solve. Each metric quantifies a different aspect of the semantic gap that raw validity conceals.

9.2.1 Pivot Preservation Rate

The most direct diagnostic: did the compressed solve keep the same pivot?

Definition 9.3 (Pivot preservation rate)**.** Given a collection of instances, let V denote the subset of instances for which the compressed solve is valid. The **pivot preservation rate** is

$$\text{PPR} \;=\; \frac{\left|\left\{i \in V :\; \tau_{\text{comp}}^{(i)} = \tau_{\text{full}}^{(i)}\right\}\right|}{|V|}.$$

When PPR $= 1.0$, every valid output preserves the original pivot. The compressed system is not merely producing well-formed output; it is producing the *same* output. When PPR < 1.0, some fraction of valid outputs have silently substituted a different pivot. The gap $1 - \text{PPR}$ is the *silent substitution rate*: the fraction of nominally successful outputs that are, in fact, telling a different story.

9.2.2 Fixed-Pivot Feasibility

Pivot preservation asks whether the solver *chose* to keep the original pivot. Fixed-pivot feasibility asks whether the solver *could have* kept it.

Definition 9.4 (Fixed-pivot feasibility)**.** The **fixed-pivot feasibility rate** is the fraction of instances for which the compressed solve is valid when forced to use the full-sequence pivot τ_{full}:

$$\text{FPF} \;=\; \frac{\left|\left\{i :\; \text{compressed solve is valid under the constraint } \tau = \tau_{\text{full}}^{(i)}\right\}\right|}{|\text{all instances}|}.$$

When FPF equals the raw validity rate, every valid compressed solution could have used the original pivot. When FPF falls below raw validity, the gap

$$\Delta_{\text{reliance}} \;=\; \text{raw validity} - \text{FPF}$$

quantifies how many instances are only valid *because* they substituted a different pivot. These instances rely on pivot substitution for their feasibility: force the original pivot, and they become infeasible.

Remark 9.5 (Relationship to the absorbing ideal). An instance with FPF $= 0$ (infeasible under the original pivot) is one in which compression has pushed the committed context element into the absorbing ideal of Proposition 7.8. The algebra predicts exactly this: once $d_{\text{pre}} < k$ under the committed pivot, no suffix can rescue the sequence. Fixed-pivot feasibility is the empirical counterpart of the absorbing predicate.

9.2.3 Semantic Regret

Pivot preservation detects substitution; fixed-pivot feasibility measures how many instances depend on it. Semantic regret quantifies the *cost* of the substitution.

Definition 9.6 (Semantic regret)**.** For an instance in which the compressed solve uses a substitute pivot, the **semantic regret** is

$$\mathrm{SR} \;=\; 1 \;-\; \frac{\mathrm{score}(\mathrm{compressed})}{\mathrm{score}(\mathrm{full})},$$

where $\mathrm{score}(\cdot)$ is the quality scoring function (a weighted combination of pivot weight and development richness).

When $\mathrm{SR} = 0$, the substitute pivot is exactly as good as the original—a benign substitution. When $\mathrm{SR} > 0$, quality has been lost. A semantic regret of 0.544 means 54.4% of the original quality has been silently discarded. The output is valid, it passes every grammar check, and it has lost more than half its meaning.

Remark 9.7 (Semantic regret is not symmetric). Semantic regret is defined as a relative shortfall from the full-sequence score. It is always non-negative when the full-sequence solve is optimal (which it is, by construction, since it searches the complete event pool). In pathological cases where the compressed solve accidentally finds a *better* pivot than the full solve—possible only under non-deterministic scoring—the regret would be negative. In our experimental framework this does not occur: the full-sequence solve is deterministic and globally optimal.

Remark 9.8 (Streaming caveat for semantic regret). Semantic regret in the streaming setting (Chapter 8) is computed under a commit-now policy: labels are assigned irrevocably as each event arrives. Deferred-commitment policies—which withhold label assignment until a fraction f of the sequence has been observed—would show different regret profiles, typically lower regret at the cost of higher latency. The regret values reported in this chapter's retention sweep apply to the batch (offline) setting and should not be compared directly with streaming regret without controlling for commitment policy.

9.3 The Retention Sweep

We now apply the three diagnostics to a controlled compression experiment. The setup is the turning-point–conditioned retention sweep from Paper 03 [5]: $n = 200$ random event sequences, prefix requirement $k = 3$, enumerative solver with beam width $M = 10$. At each retention level, the event sequence is compressed by randomly removing non-focal events until the target retention fraction is reached, and the solver is applied to the compressed sequence. The results are presented in Table 9.1.

The pattern in Table 9.1 is the empirical signature of the validity mirage. Consider the trajectory of each column.

Raw validity. The first column is monotonically near-perfect. At 90% retention, all 200 instances produce valid output. At 50%, still all 200. At 30%, still all 200. Even at 10% retention—where 90% of the non-focal events have been removed—validity drops only to 0.990. A practitioner monitoring this column alone would conclude that compression is essentially lossless.

Table 9.1: The validity mirage across retention levels ($n = 200$, $k = 3$, $M = 10$). Raw validity stays at or near 1.0 across all retention levels, while pivot preservation collapses from 0.790 at 50% retention to 0.354 at 10% retention. The mirage gap—the difference between raw validity and pivot preservation—widens to ≈ 0.64 at 10% retention. Retention levels 0.90 and 0.70 were not swept in the original paper; see the footnote on those rows.

Retention	Raw Validity	Pivot Preservation	Fixed-Pivot Feas.	Semantic Regret
0.90	—[1]	—	—	—
0.70	—	—	—	—
0.50	1.000	0.790	0.980	0.218
0.30	1.000	0.590	0.920	0.663
0.20	1.000	0.480	0.860	0.677
0.10	0.990	0.354	0.750	0.358

Pivot preservation. The second column tells a different story. At 90% retention, nearly every valid output preserves the original pivot. By 50% retention, only 79% of valid outputs keep the same pivot—one in five has silently substituted. By 30%, only 59% preserve the original; by 10%, only 35.4%. At 10% retention, the system substitutes the pivot in nearly two out of every three valid outputs.

Fixed-pivot feasibility. The third column reveals why the pivots are being substituted. At 50% retention, 98% of instances are still feasible under the original pivot—so the 21% of outputs that substituted pivots *could have* kept the original, but the enumerative solver found an alternative path. By 10% retention, only 75% of instances remain feasible under the original pivot. The remaining 25% *cannot* be solved with the original pivot at all: compression has pushed them into the absorbing ideal.

Semantic regret. The fourth column quantifies the cost. At 50% retention, the average quality loss among substituted instances is 21.8%. At 30%, it peaks at 66.3%. At 10%, it settles at 35.8%—not because substitution has become less harmful, but because the most severely affected instances have been pushed into infeasibility entirely, leaving a survivorship-biased sample.

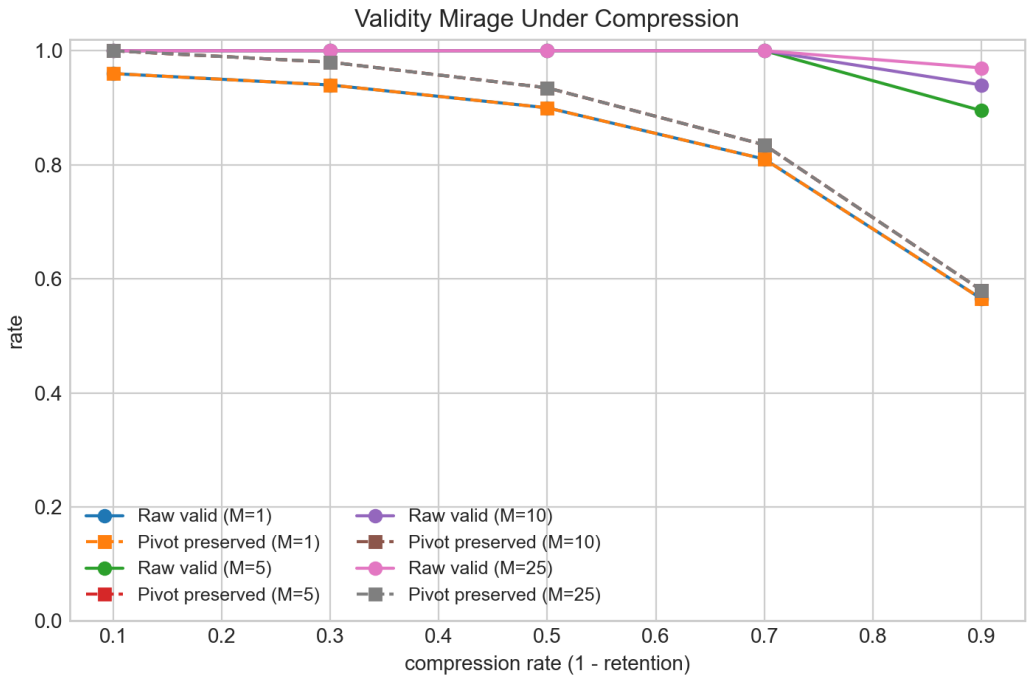


Figure 9.1: The validity mirage: raw validity (blue) stays near 1.0 while pivot preservation (red) collapses under increasing compression. The widening gap is the mirage—where the system reports success but has silently substituted the semantic anchor.

Figure 9.1 visualises the divergence. The blue curve (raw validity) hugs the ceiling. The red curve (pivot preservation) falls away. The vertical distance between the two curves at any retention level is the *mirage gap*: the fraction of outputs that are valid but semantically unfaithful. At 10% retention, the mirage gap is $0.990 - 0.354 = 0.636$—nearly two-thirds of all outputs are mirages.

9.4 The Deterministic Witness

The retention sweep demonstrates the mirage statistically. This section constructs a single, deterministic instance that exhibits all three mirage conditions simultaneously. The witness is the most important concrete example in the book: it makes the mirage mechanism fully transparent by isolating it in a sequence small enough to trace by hand.

9.4.1 Witness Construction

The witness is produced by the function `deterministic_mirage_witness(k=3)` in `src/compression.py`. It constructs a specific event sequence designed to satisfy two properties:

(i) The full sequence has a unique dominant pivot with high weight and sufficient pre-pivot development.

(ii) Compression removes enough non-focal events to make the dominant pivot infeasible under committed semantics, while leaving an alternative pivot feasible under enumerative search.

The resulting sequence exhibits the following triple of outcomes, which we now examine in detail.

9.4.2 The Three Solves

Solve 1: Full sequence (uncompressed). The full event sequence is processed by the solver with no compression. The dominant pivot is event 4 (the fifth event in the 10-event sequence), with weight $w^\star = 20.0$. Because event 4 appears with sufficient non-focal events preceding it, $d_{\text{pre}} \geq k = 3$. The solve is **feasible**, the quality score is 20.0, and this is the reference solution against which all compressed solves are compared.[2]

Solve 2: Naive compressed, committed ($M = 1$). The same sequence is compressed by removing non-focal events (naive random compression, no contract guard). The committed solver with $M = 1$ is required to use the original pivot (event 4). After compression, the non-focal events that preceded event 4 have been thinned: d_{pre} drops below k. Since $M = 1$, the solver has no alternative candidates to consider. The result is **infeasible**.

This is the absorbing ideal at work. The committed context element has $\kappa = 1$ and $d_{\text{pre}} < k$, placing it in the left ideal of Proposition 7.8. No suffix can rescue it.

Solve 3: Naive compressed, enumerative ($M = 10$). The same compressed sequence, but now with the enumerative solver ($M = 10$ pivot candidates). The solver discovers that event 4 is infeasible and begins searching alternatives. It finds event 8 (the substitute pivot): a focal event with lower weight $w^\star = 13.0$, appearing at a later position in the sequence, with a different set of preceding non-focal events such that $d_{\text{pre}} \geq k = 3$ under the compressed sequence. The solve is **feasible**.

But the quality score under the substitute pivot is only 13.0. The semantic regret is

$$\text{SR} \;=\; 1 - \frac{13.0}{20.0} \;=\; 0.35.$$

The system reports a valid output. The grammar is satisfied. The phase labels are well-formed. And 35% of the semantic quality has been silently discarded. The output tells a different story—one anchored at a weaker event with less dramatic weight—and no signal in the system's output reveals this fact.

9.4.3 Mechanism Diagram

The three solves correspond to three rows in the mechanism diagram (Figure 9.2). Each row represents the same underlying event sequence under a different solver configuration.

[2]The book's deterministic witness uses a 10-event sequence; the paper (Table 8) used a larger 50-event synthetic instance with correspondingly higher indices.

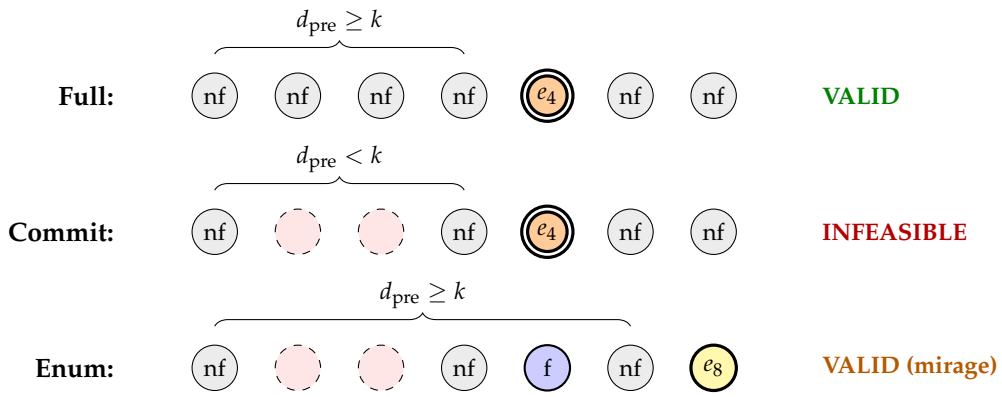


Figure 9.2: Mechanism diagram for the deterministic witness. **Row 1:** The full sequence places the dominant pivot e_4 (orange, double border) with sufficient pre-pivot development ($d_{\text{pre}} \geq k$). Valid, high quality. **Row 2:** Compression removes non-focal events from the middle (dashed red). The committed solver ($M = 1$) retains e_4 as pivot, but d_{pre} drops below k. Infeasible: the absorbing ideal has captured this state. **Row 3:** The enumerative solver ($M = 10$) searches alternatives and finds e_8 (yellow), a weaker pivot with a different set of preceding events satisfying $d_{\text{pre}} \geq k$. Feasible—but with a different pivot and 35% semantic regret.

The three rows of Figure 9.2 make the mechanism visually explicit. In Row 1, the full sequence supports the dominant pivot with ample pre-pivot development. In Row 2, compression thins the pre-pivot region, and the committed solver has no recourse—it is trapped in the absorbing ideal. In Row 3, the enumerative solver escapes absorption by shifting to a weaker pivot, exactly as predicted by Remark 7.10: under $\otimes_{\text{endo}}$ semantics, a suffix with a higher d_{pre} count can escape the absorbing set by adopting a different pivot. The escape is real—the output is valid—but the semantic cost is 35%.

9.4.4 Witness Results

Table 9.2 presents the complete results for the deterministic witness across all four solver configurations: naive and contract-guarded compression, each with $M = 1$ (committed) and $M = 10$ (enumerative).

Table 9.2: Deterministic witness results (Experiment 58). The naive solver either fails ($M = 1$) or substitutes ($M = 10$). The contract-guarded solver preserves the original pivot in all configurations.

Strategy	M	Free Valid	Free TP	Fixed Valid	Preserved
Naive	1	False	—	False	False
Contract	1	True	e_4	True	True
Naive	10	True	e_8	False	False
Contract	10	True	e_4	True	True

The table should be read row by row.

- **Naive, $M = 1$:** The committed solver finds the original pivot e_4 infeasible and has no alternatives. Both free and fixed-pivot solves fail. This is an outright failure, not a mirage—the system correctly reports infeasibility.

- **Contract, $M = 1$:** The contract-guarded compression preserves enough development events around e_4 to maintain $d_{\text{pre}} \geq k$. The committed solver succeeds with the original pivot. No substitution, no quality loss.
- **Naive, $M = 10$:** The enumerative solver discovers that e_4 is infeasible and substitutes e_8. The free solve succeeds (valid output), but the fixed-pivot solve fails (cannot use e_4), and pivot preservation is `False`. This is the mirage: valid output, wrong pivot.
- **Contract, $M = 10$:** Contract-guarded compression again preserves the original pivot. The enumerative solver does not need to search alternatives because e_4 remains feasible. Preservation is `True`.

The witness demonstrates that the contract of Definition 7.13 is not merely a theoretical safeguard: it is the operational mechanism that prevents the mirage. Under naive compression, the system either fails or lies (substitutes silently). Under contract-guarded compression, it tells the truth.

9.5 Taxonomy of Mirages

Not all semantic failures present identically. We distinguish three types, ordered by increasing difficulty of detection.

9.5.1 Silent Mirage

Definition 9.9 (Silent mirage)**.** A **silent mirage** is an instance satisfying all four conditions of Definition 9.1. The system produces valid output with a substituted pivot. No error, no warning, no degraded-evidence flag is raised. The user cannot distinguish the mirage from a genuine success using only the system's output.

The silent mirage is the most dangerous type precisely because it is invisible. The deterministic witness of Section 9.4 (naive compression, $M = 10$) is a silent mirage: the output is valid, the grammar is satisfied, and no signal indicates that the pivot has changed. The only way to detect it is to compare against the full-sequence solve—an external reference that the system does not provide.

9.5.2 Protocol Collapse

Definition 9.10 (Protocol collapse)**.** A **protocol collapse** occurs when the model stops emitting required structural elements entirely. Unlike a silent mirage, protocol collapse produces *invalid* output—but the invalidity may be partial, passing looser validation checks while failing stricter ones.

Protocol collapse is qualitatively different from the silent mirage. In a silent mirage, the output is fully valid under the grammar; the failure is semantic, not structural. In a protocol collapse, the output is structurally deficient: a required phase is missing, a mandatory field is empty, a schema element has been dropped. A typical example is a summariser that drops the conclusion section under context pressure, or a report generator that omits mandatory headers when the input is compressed.

Protocol collapse is easier to detect than a silent mirage because it violates the grammar. However, if the validation check is permissive—accepting partial outputs, or checking only a subset of required elements—the collapse may pass unnoticed.

9.5.3 Representation-Level Mirage

Definition 9.11 (Representation-level mirage)**.** A **representation-level mirage** occurs in neural systems when the evidence for the correct pivot is present in the input tokens but becomes inaccessible after internal compression—typically KV-cache eviction [25, 27] in transformer models. The model "knows" the information was there but can no longer attend to it.

The representation-level mirage is the analogue of the algebraic mirage lifted to the attention mechanism of a language model. In our framework, KV-cache eviction is a compression map μ that operates not on the event sequence directly but on the model's internal representation of that sequence. The tokens encoding the dominant pivot may survive eviction, but the attention weights connecting those tokens to their supporting context—the pre-pivot development events—are lost. The model retains the pivot's identity but loses the ability to verify or utilise the pivot's structural support.

Table 9.3 presents KV-cache eviction results on Llama 3.1 8B, demonstrating the representation-level mirage across retention levels.

Table 9.3: Representation-level mirage under KV-cache eviction (Llama 3.1 8B). Header compliance and pivot preservation both degrade under eviction, while raw validity remains relatively stable—the model produces grammatically acceptable output even when it can no longer access the correct pivot.

Retention	Header Compliance	Pivot Preserve	Raw Validity	Semantic Regret
1.0	0.917	1.000	0.962	0.000
0.7	0.500	0.583	0.703	0.320
0.5	0.833	0.500	0.508	0.375
0.3	0.417	0.167	0.629	0.375
0.1	0.667	0.083	0.680	0.330

Several features of Table 9.3 deserve attention.

Pivot preservation collapses. At full retention, pivot preservation is 1.000: the model always identifies the correct turning point. At 70% retention, it drops to 0.583. At 10% retention, only 8.3% of outputs preserve the correct pivot. The model is almost never telling the right story.

Raw validity remains relatively high. Even at 10% retention, raw validity is 0.680—the model produces grammatically acceptable output in more than two-thirds of cases. The mirage gap at 10% retention is $0.680 - 0.083 = 0.597$: nearly 60% of outputs are valid mirages.

Header compliance is non-monotone. Header compliance drops from 0.917 to 0.417 at 30% retention but then recovers to 0.667 at 10%. This non-monotonicity suggests that at extreme compression levels, the model defaults to a formulaic output pattern that happens to include headers—a form of protocol compliance without semantic content.

9.6 External Validation

The preceding experiments use synthetic event sequences. To confirm that the validity mirage is not an artefact of the synthetic generator, we validate on two external benchmarks: real-incident event graphs and a multi-model blackbox sweep.

9.6.1 NTSB Real-Incident Graphs

Twelve real aviation and financial incidents were encoded as event graphs using the same structural format as the synthetic generator. Each incident graph was processed under both naive and contract-guarded compression across multiple retention levels, producing 164 total instance–retention combinations per compression strategy.

Proposition 9.12 (Silent mirage rates on real incidents)**.** *Under naive compression,* 36 *of* 164 *instance–retention combinations exhibited a silent mirage, for a rate of* 21.95%*. Under contract-guarded compression,* 0 *of* 164 *combinations exhibited a silent mirage, for a rate of* 0%*.*

At matched retention (budget 0.7), the silent mirage rate for naive compression is 23.5%; for contract-guarded compression, it remains 0%. The contract is not merely effective on synthetic data: it eliminates the mirage entirely on real incidents drawn from safety-critical domains.

9.6.2 Multi-Model Blackbox Sweep

To verify that the mirage is not specific to any one model architecture, five language models were tested in a blackbox configuration: Llama 3.1 8B [11], Mistral 7B [13], Gemma 2 9B [9], Phi-3 Medium 14B [1], and Qwen 2.5 14B [26]. Each model was given compressed event sequences and asked to produce structured output (narrative arcs with phase labels). The diagnostic metrics were computed by comparing each model's output against the full-sequence reference.

All five architectures exhibit the same qualitative pattern: raw validity remains high while pivot preservation degrades under compression. The mirage is architecture-independent. It arises not from model-specific weaknesses but from the structural interaction between compression and endogenous pivot selection—the same mechanism that the algebra predicts.

One notable finding is that the *investment* category is the fragile basin across models. Investment-related incidents show approximately 20% pivot preservation under compression, compared to 80% for general incident categories. This fragility reflects the structural properties of the category: investment event graphs tend to have many candidate pivots of similar weight, making the dominant pivot easy to displace. The incident category, with its more sharply differentiated weight distribution, is more robust to compression.

9.7 The Algebraic Explanation

The empirical results of this chapter are not merely empirical: they are predicted by the algebraic theory of Chapter 7. We now close the loop by connecting each observation back to its algebraic cause.

9.7.1 Committed Absorption Is Permanent

Under $\otimes_{\text{commit}}$ (committed semantics), absorbed states are permanent. This is Proposition 7.8: if $\kappa = 1$ and $d_{\text{pre}} < k$, then for every suffix $\bar{C}_D$,

$$\bar{C}_A \otimes_{\text{commit}} \bar{C}_D \in \bot.$$

The committed context element is trapped in the absorbing ideal. No continuation can rescue it. This is exactly what the deterministic witness demonstrates in Solve 2: the committed solver with $M = 1$ cannot recover from the compression-induced prefix deficiency.

9.7.2 Endogenous Escape Routes

Under $\otimes_{\text{endo}}$ (endogenous semantics), a suffix with a higher-weight pivot can escape absorption. This is the mechanism described in Remark 7.10 and Example 7.11: if the suffix contains a pivot with $w^\star{}_D > w^\star{}_A$ and sufficient pre-pivot development, the composite element escapes the absorbing set by shifting to the new pivot.

The enumerative solver with $M > 1$ exploits this mechanism. When the original pivot is infeasible, the solver searches for alternative pivots—effectively exploring $\otimes_{\text{endo}}$ escape routes from the absorbing set. The search succeeds whenever there exists a candidate pivot with sufficient pre-pivot development in the compressed sequence. The resulting output is valid, but the pivot identity has changed. This is the algebraic origin of the mirage: the gap between committed and endogenous feasibility.

The committed semantics say: "this state is broken; the absorbing ideal has captured it." The endogenous semantics say: "I can fix it by using a different pivot." The fix is real—the output is grammatically valid—but the meaning has changed. The mirage is precisely this gap: the system escapes structural failure by abandoning semantic fidelity.

9.7.3 Compression Is the Unique Closure-Breaking Operation

Section 7.7.3 (Experiment 57) established that compression is the unique operation that violates algebraic closure. The violation rates bear repeating in the present context:

Operation	Violation rate
Composition ($\otimes_{\text{commit}}$)	0.000
Compression	0.133
Pivot update	0.000
Split-at-point	0.000

Every algebraic operation except compression preserves the monoid structure perfectly. Composition, pivot update, and split-at-point all have zero violation rates. Only compression can push elements across the absorbing boundary—at a rate of 13.3% when unguarded.

This result explains why the mirage is specifically a compression pathology. If the system only composed, split, and updated pivots, the monoid structure would be preserved and no element would cross the absorbing boundary involuntarily. Compression is the singular point of vulnerability, and the no-absorption contract of Definition 7.13 is the precisely targeted guard.

9.7.4 The Mirage as a Gap Between Two Semantics

We can now state the algebraic characterisation of the validity mirage concisely.

Remark 9.13 (Algebraic characterisation of the mirage). The validity mirage is the gap between committed and endogenous feasibility after compression. Let $\bar{C}$ be the context element of a compressed sequence. Under $\otimes_{\text{commit}}$, the original pivot may be infeasible ($\bar{C} \in \bot$). Under $\otimes_{\text{endo}}$, a substitute pivot may restore feasibility ($\bar{C} \notin \bot$ after pivot shift). The mirage exists whenever the committed solve is infeasible but the endogenous solve is feasible—or, more subtly, whenever the endogenous solve is feasible with a *different* pivot than the committed solve would have chosen.

The three diagnostics of Section 9.2 measure three aspects of this gap:

- **Pivot preservation** measures how often the gap is zero (no substitution).
- **Fixed-pivot feasibility** measures how often committed feasibility survives compression (no absorption).
- **Semantic regret** measures how much quality the endogenous escape costs when the gap is nonzero.

9.8 Exercises

Exercise 9.1 (Computing the mirage gap)**.** Using the data from Table 9.1, compute the mirage gap Δ_{mirage} at each retention level. At which retention level is the mirage gap largest? Explain why the mirage gap at 10% retention is smaller than the gap at 20% retention, despite the compression being more aggressive.

Exercise 9.2 (Constructing a mirage-free witness)**.** Construct an event sequence with $k = 3$ and at least 10 events such that naive compression to 50% retention *cannot* produce a silent mirage, regardless of which events are removed. *Hint:* Consider a sequence in which every focal event has the same weight and sufficient pre-pivot development. Why does equal weighting prevent silent substitution?

Exercise 9.3 (Fixed-pivot feasibility and the absorbing predicate)**.** Prove that for a compressed instance with committed context element $\bar{C} = (w^\star, d_{\text{total}}, d_{\text{pre}}, 1)$ and prefix requirement k, the following are equivalent:

(a) The fixed-pivot solve is infeasible.

(b) $\bar{C} \in \bot$ (i.e., $d_{\text{pre}} < k$).

That is, fixed-pivot infeasibility is exactly the absorbing predicate applied to the committed context element.

Exercise 9.4 (Semantic regret under survivorship bias)**.** Explain the non-monotone behaviour of semantic regret in Table 9.1: why does semantic regret *decrease* from 0.677 at 20% retention to 0.358 at 10% retention, even though compression has become more aggressive? Your explanation should invoke the concept of survivorship bias. *Hint:* Which instances contribute to the semantic regret average at 10% retention? What happened to the instances with the highest potential regret?

Exercise 9.5 (The representation-level mirage and attention)**.** Consider a transformer model with KV-cache eviction at 30% retention. The model has 8 attention heads and a context window of 4096 tokens. Suppose the dominant pivot is encoded at token position 3500, and the $k = 3$ pre-pivot development events are encoded at positions 1200, 2100, and 3200.

(a) If eviction removes all tokens with position > 2048, which development events survive? Is the pivot still feasible?

(b) If eviction uses a "recent window" policy (keeping the most recent 30% of tokens), which tokens survive? Does this policy preserve pivot feasibility better than the positional cutoff?

(c) Why does the non-monotone header compliance in Table 9.3 suggest that aggressive eviction causes the model to fall back on memorised templates rather than attending to context?

Chapter 10

The Narrative Origin Story

The formal theory developed in Chapters 3, 5, 7 and 9 was not built in the abstract. It grew out of a specific system—the Lorien narrative simulation [3], a story-sifting system [8]—and a specific set of failures that resisted every ad hoc fix we tried. This chapter returns to the origin. We present the three experimental findings that motivated the algebraic programme, show how each finding connects to a theorem or construction from the preceding chapters, and demonstrate that the formal theory retroactively explains every empirical anomaly we observed.

The structure follows three questions, each answered by one section. Section 10.1 asks: *why does a stricter grammar produce better arcs?* Section 10.2 asks: *why does Diana's arc collapse in exactly 9 out of 50 seeds?* Section 10.3 asks: *why does agent evolution help in depleted environments but hurt in fresh ones?* A final section (Section 10.4) maps each finding to the formal apparatus.

10.1 Grammar as Regularizer

The Lorien arc-extraction pipeline scores candidate arcs with a *quality metric* Q that combines tension, irony, and thematic coherence into a single scalar. The pipeline then selects the highest-Q arc that satisfies a *beat grammar*—the monotonic phase grammar of Section 3.2, with four phases in strict order:

$$\text{SETUP} \longrightarrow \text{DEVELOPMENT} \longrightarrow \text{TURNING\_POINT} \longrightarrow \text{RESOLUTION}.$$

The grammar enforces three structural constraints: (i) no phase regressions (monotonicity), (ii) at most a bounded number of turning points, and (iii) at least $k \geq 1$ development beats before the turning point. We call this the *strict grammar*.

A natural question arose during development: is the grammar unnecessarily restrictive? Would relaxing it allow the Q-metric to find higher-quality arcs? We tested a *relaxed grammar* that permitted 1–2 turning points and allowed up to one phase regression. The relaxed grammar is strictly more permissive: every strict-valid arc is also relaxed-valid. The results were unambiguous and surprising.

The strict grammar achieves an 88% all-valid rate across seeds. The relaxed grammar—which admits every sequence the strict grammar admits, plus many more—collapses to 32%. Permissiveness destroyed performance.

Metric	Strict grammar	Relaxed grammar
All-valid rate	88%	32%

Table 10.1: Grammar relaxation experiment. The relaxed grammar is strictly more permissive than the strict grammar, yet the all-valid rate collapses from 88% to 32%.

10.1.1 Why Permissiveness Hurts

The explanation is that the strict grammar acts as a *regularizer* during search. With strict constraints, the search algorithm is forced to select events that form a coherent narrative arc: setup events precede development events, development events precede the turning point, and the turning point precedes the resolution. Each constraint prunes the search space, channelling the Q-maximising selection toward sequences that are not merely high-scoring but narratively well-formed.

Without strict constraints, the Q-metric is free to select high-scoring *fragments* that are narratively incoherent. An early catastrophe followed by sixteen consequence events can score well on tension and irony—the catastrophe is tense, and the consequences provide thematic resonance—while lacking the developmental arc that makes a story function as a story. The grammar prevents precisely this failure: it is a structural guard against Goodhart's law [10], in which a measure that becomes a target ceases to be a good measure—a phenomenon studied more broadly as reward hacking [22]. When Q is the only objective, the search exploits Q at the expense of narrative structure. When the grammar co-constrains the search, the search cannot exploit Q without also satisfying structural requirements.

10.1.2 Single-Constraint Attribution

To identify which grammatical constraint carries the regularisation effect, we relaxed each of the four constraint dimensions independently:

(i) **Minimum development beats** (`min_development_beats`): relaxed from ≥ 1 to ≥ 0.

(ii) **Maximum phase regressions** (`max_phase_regressions`): relaxed from 0 to 1.

(iii) **Protagonist coverage** (`protagonist_coverage`): relaxed threshold.

(iv) **Minimum timespan fraction** (`min_timespan_fraction`): relaxed threshold.

Only `min_development_beats` matters. When we relax it from requiring ≥ 1 to requiring ≥ 0 development beats, the all-valid rate jumps from 64% to 100%. The other three dimensions are *structurally inert*: relaxing any of them produces no change in validity. The regularisation effect is concentrated entirely in the development-beat requirement.

10.1.3 The Rescore-Only Test

A rescore-only test confirms that the failure is one of *content absence*, not constraint mismatch. We took all 855 strict-valid sequences from the experimental corpus and validated them against the relaxed grammar: 855 out of 855 passed (100%). We then took the 45 strict-invalid sequences and validated them against the relaxed grammar: 0 out of 45 recovered.

If the failure were a constraint mismatch—sequences that are structurally sound but happen to violate a technicality of the strict grammar—then rescoring against the relaxed grammar would

Source sequences	Count	Relaxed-valid
Strict-valid	855	855/855 (100%)
Strict-invalid	45	0/45 (0%)

Table 10.2: Rescore-only test. Strict-valid sequences universally pass the relaxed grammar. Strict-invalid sequences universally fail it. The failure is content absence, not constraint mismatch.

Metric	Valid ($N = 41$)	Invalid ($N = 9$)
Simulation involvement	43.5	42.7
Arc events	20.0	20.0
Complication beats	3.3	0.0
Escalation beats	2.9	0.0
Consequence beats	7.8	16.4
TP position (median)	0.69	0.13
Events before TP	11.2	2.6
Events after TP	7.8	16.4

Table 10.3: Diagnostic comparison of Diana's valid and invalid arcs across 50 seeds. Invalid arcs have comparable simulation involvement and identical arc length, but zero development beats, extremely early turning-point position, and a heavily consequence-dominated structure.

recover at least some of them. The fact that zero recover means the sequences lack development content entirely. There are no development events to score, under any grammar.

Remark 10.1 (Endogenous pivot connection). This finding is a concrete instance of the endogenous pivot problem. The search selects events that maximise Q. Without the development requirement, Q-maximising selections skip development events entirely, anchoring the turning point early—typically on the highest-tension event, which is often an early catastrophe—and filling the arc with high-tension consequences. The grammar forces the search to include development events, which pushes the turning point later in the timeline and creates a proper dramatic arc. The strict grammar does not merely filter bad arcs; it reshapes the search landscape so that the Q-optimal arc within the constrained space is qualitatively different from the Q-optimal arc in the unconstrained space.

10.2 Phase Collapse Anatomy: Diana's Arc

Chapter 1 introduced Diana's collapsed arc as a motivating example. We now dissect the failure in full quantitative detail.

Diana is a peripheral observer (evader archetype) in a six-agent dinner-party simulation. Of 50 random seeds under full agent evolution, 9 produce invalid arcs for Diana. All 9 fail identically: zero complication or escalation beats, meaning the DEVELOPMENT phase is completely absent.

10.2.1 Diana Is Not Event-Starved

A natural hypothesis is that Diana's failures reflect insufficient simulation material—perhaps she simply does not participate in enough events to form an arc. The data refute this hypothesis decisively. Table 10.3 presents the key diagnostic comparison.

Diana's mean simulation involvement in the 9 invalid seeds is 42.7 events—virtually identical to the 43.5-event mean in valid seeds. The arc lengths are the same: 20.0 events in both cases. The event pool is present; the events are simply the wrong kind. In invalid arcs, the entire developmental middle of the story—complication and escalation beats—is absent. In its place, 16.4 consequence beats fill the arc after a turning point that arrives at normalised position 0.13.

10.2.2 Turning-Point Anchoring

The turning-point position data reveal a categorical separation, not a gradual degradation. Valid arcs have a median turning-point position of 0.69—the classical mid-to-late position that leaves room for both development and resolution. Invalid arcs have a median position of 0.13—barely past the opening.

The distributions have *zero overlap*. No valid arc has a turning point as early as the latest invalid turning point; no invalid arc has a turning point as late as the earliest valid one. This is bimodal failure: two categorically distinct regimes, not a continuum of degradation.

10.2.3 Candidate Pool Contamination

The mechanism becomes clear when we examine the candidate pools. Across the 9 invalid seeds, there are 33 total candidate arcs. Every single one has zero development beats. Turning-point positions range from 0.062 to 0.161. The pools are *entirely degenerate*—every candidate produces the same early-TP, no-development failure.

In contrast, the valid seeds contain 38 candidates, of which 36 (94.7%) have at least one development beat. The valid pools are overwhelmingly healthy; the invalid pools are uniformly broken.

10.2.4 The Dual-Function Injection Mechanism

The Lorien pipeline includes a *protagonist-event injection* step: for peripheral agents like Diana, the system adds the focal actor's maximum-weight event to the candidate pool. This injection is structurally necessary—without it, Diana's solo breadth-first search yields 0 out of 50 valid arcs, because she has too few directly-connected events to form a complete arc.

But the injection has a side effect. The injected events are high-weight, early-timeline events from the simulation's dramatic opening: confrontations, revelations, catastrophes that drove the main plot. These events contaminate the candidate pool with early pivots. The greedy search, which selects the turning point as the highest-weight focal event, locks onto these early catastrophes. With the turning point anchored before position 0.20, there is almost no timeline available for development beats, and the grammar's prefix requirement becomes unsatisfiable.

Temporal injection filter. We tested this diagnosis by excluding injection events that occur before normalised position 0.20 in the timeline. The filter recovers 7 of the 9 focal seeds, raising the all-valid rate from 41/50 to 47/50. This confirms the mechanism: early-event contamination via the injection step is the root cause of the collapse.

10.2.5 Two-Regime Classification

The 9 failures decompose into two distinct regimes:

(i) **Search exploration failure (5/9 seeds).** Valid mid-arc alternatives exist that outscore the search-selected invalid arc. Better arcs are present in the candidate space, but the greedy search does not find them. The failure is in search coverage, not in the metric or the event pool.

(ii) **Metric misalignment (4/9 seeds).** The invalid arc actually outscores the best valid alternative by approximately 0.098 in Q. The quality metric genuinely prefers the early kinetic spike—the high-tension catastrophe followed by rapid consequences—over a structurally sound arc with proper development. The failure is in the metric itself.

Despite this decomposition, both regimes share a single root cause: premature turning-point anchoring via pool contamination. In regime (i), the contamination blinds the search; in regime (ii), the contamination biases the metric. The upstream mechanism is identical.

10.3 Evolution as Pacing Control

The Lorien system evolves agent profiles across simulation depth: after each simulation run, an agent's personality parameters are updated based on the events it experienced. This section examines the interaction between evolution and arc validity.

10.3.1 The Quality–Validity Tradeoff

We measure two metrics as a function of *coalition size* c, the number of agents whose profiles are evolved (holding all others at their default profiles):

- **Mean** Q: the raw quality score, averaging tension, irony, and thematic coherence.
- **VA** (validity-adjusted score): Q scaled by the fraction of arcs that are valid.

The two metrics diverge systematically. Mean Q increases monotonically with coalition size: evolving more agents produces richer, more dramatic simulations. But VA drops from its $c = 0$ value of 0.652 to a minimum of 0.617 at $c = 3$ before partially recovering. The all-valid rate drops from 88% at $c = 0$ (no evolution) to 64% at $c = 6$ (full evolution).

Evolution makes the simulation more interesting but makes the arcs harder to extract. Evolved agents produce higher-tension events that cluster early in the timeline, contaminating candidate pools and anchoring turning points prematurely—the same mechanism we identified in Diana's collapse.

10.3.2 The α-Interpolation Experiment

To study the evolution effect at finer resolution, we interpolate between default and evolved agent profiles using a blending parameter $\alpha \in [0, 1]$:

$$\theta(\alpha) \;=\; (1-\alpha)\,\theta_{\text{default}} \;+\; \alpha\,\theta_{\text{evolved}}.$$

For Thorne—the destabilising agent whose evolution most strongly affects arc validity—VA is sharply peaked at $\alpha = 0.5$, where it reaches 0.684 with a 95% all-valid rate. The optimum is driven entirely by validity, not raw quality: Q is essentially flat across α. The peak at $\alpha = 0.5$ reflects a balance point where Thorne is evolved enough to generate rich events but not so evolved that his early-timeline catastrophes overwhelm the candidate pools.

10.3.3 Factorial Decomposition

A 2×3 factorial design (2 simulation depths $\times$ 3 evolution levels) decomposes the evolution effect into three orthogonal components:

(i) **Amplifier effect** (depth D_0, default $\rightarrow$ evolved): $+0.008$ mean Q, -0.028 VA. Evolution *hurts* validity in fresh environments.

(ii) **Degradation effect** ($D_0 \rightarrow D_2$, default profiles): -0.007 mean Q, -0.009 VA. Information depletion across simulation depth reduces quality.

(iii) **Repair effect** (depth D_2, default $\rightarrow$ evolved): $+0.016$ mean Q, $+0.013$ VA. Evolution *helps* in depleted environments.

The repair effect exceeds the amplifier effect by 4 VA points. Evolution is primarily a *repair mechanism* for depleted environments, not an improvement mechanism for fresh ones. In a fresh simulation (D_0), the event landscape is rich enough that the arc extractor can find valid arcs without assistance; evolution merely adds high-tension events that contaminate the pools. In a depleted simulation (D_2), the event landscape has been thinned by repeated extraction; evolution replenishes the pool with new material, restoring the development capacity that depletion removed.

Remark 10.2 (Scaffolding dependence). The depleted canon provides structural scaffolding that constrains agent behaviour into extractable patterns. Evolution *needs* that scaffolding to work. In a fresh environment, the agents have no history to constrain them, and evolved profiles produce unconstrained high-tension events that are structurally unmoored. In a depleted environment, the accumulated narrative history channels evolved behaviour into patterns that the arc extractor can recognise and extract. The scaffolding converts evolution from a destabilising amplifier into a targeted repair mechanism.

10.4 Connection to the Formal Theory

Each of the three empirical findings maps directly to a construction or theorem from the preceding chapters. We make these connections explicit.

10.4.1 Phase Collapse and the Absorbing State Theorem

Diana's phase collapse (Section 10.2) is a concrete instance of the prefix-constraint impossibility theorem (Theorem 3.8 in Chapter 3). In Diana's invalid arcs, the development-eligible count is exactly zero: $j_{\text{dev}} = 0$. The grammar's prefix requirement is $k = 1$. Since $j_{\text{dev}} = 0 < k = 1$, Theorem 3.8 applies directly: the greedy policy produces zero valid sequences. This is not a statistical tendency or a soft failure mode—it is an exact impossibility, predicted by a counting argument, confirmed across all 9 invalid seeds without exception.

The theorem also explains *why* the failure is categorical rather than gradual. The absorbing-state boundary at $j_{\text{dev}} = k$ is a sharp threshold: below it, validity is exactly zero; above it, validity is possible. The bimodal separation in turning-point position (Section 10.2.2)—zero overlap between valid and invalid distributions—is the empirical signature of this sharp boundary. Seeds that contaminate Diana's pool with early pivots push j_{dev} below k and enter the impossibility zone. Seeds that avoid contamination leave j_{dev} well above k and land in the high-validity zone. There is no middle ground, exactly as the theorem predicts.

10.4.2 Grammar Regularization and the Absorbing Ideal

The grammar regularisation result (Section 10.1) illustrates why the absorbing ideal (Chapter 7) matters practically. The absorbing ideal characterises the algebraic boundary between feasible and infeasible context elements: an extended context element $\bar{C} = (w^\star, d_{\text{total}}, d_{\text{pre}}, \kappa)$ with $\kappa = 1$ and $d_{\text{pre}} < k$ belongs to the ideal, and no suffix can rescue it.

The grammar's development-beat requirement is the operational enforcement of the condition $d_{\text{pre}} \geq k$. When this requirement is present, the search is forced to include development events, which ensures that the selected arc's context element satisfies $d_{\text{pre}} \geq k$ and stays outside the absorbing ideal. When the requirement is relaxed to $d_{\text{pre}} \geq 0$, the search is free to select arcs whose context elements fall inside the ideal. The Q-metric, unconstrained by structural requirements, gravitates toward these ideal elements because they correspond to early-TP, consequence-heavy arcs that score well on tension.

The strict grammar does not merely filter outputs; it constrains the search to operate outside the absorbing ideal. The rescore-only test (Section 10.1.3) confirms this interpretation: strict-invalid arcs are not borderline cases that a permissive grammar could rescue. They are deep inside the ideal—$j_{\text{dev}} = 0$, not $j_{\text{dev}} = k - 1$—and no relaxation of grammatical constraints can supply the development content that is entirely absent.

10.4.3 Injection Contamination and the Validity Mirage

The injection contamination mechanism (Section 10.2.4) is a specific pathway to the validity mirage (Chapter 9). The system *could* produce a valid arc with a different pivot—in 5 of the 9 failures, valid alternatives exist that outscore the selected arc. But the greedy search locks onto the contaminating early pivot and cannot escape. From the system's perspective, the arc-extraction succeeded: it found a turning point (the highest-weight event), assembled a sequence around it, and reported the result. From a structural perspective, the result is broken.

This is precisely the mirage pattern: the system reports success on a metric (turning-point selection, sequence assembly) while concealing a structural failure (absent development, premature anchoring). The four-metric diagnostic checklist from Section 11.2—pivot preservation, fixed-pivot feasibility, semantic regret, and mirage gap—would detect this failure immediately. The contaminating early pivot fails the fixed-pivot feasibility test (there are not enough pre-pivot events to satisfy the grammar if we force the original pivot), and the semantic regret relative to a properly positioned pivot is catastrophic.

The two-regime classification (Section 10.2.5) further illuminates the mirage. In the search-exploration regime, valid arcs exist but the search misses them: this is a mirage of search coverage, where the system appears to have explored the candidate space but has in fact been trapped by the contaminated pool. In the metric-misalignment regime, the Q-metric actively prefers the broken arc: this is a mirage of metric alignment, where the quality score appears to validate the selection while concealing its structural deficiency.

Taken together, these three connections—absorbing states, the absorbing ideal, and the validity mirage—demonstrate that the formal theory is not an ex post rationalisation. The theory was built to explain these specific failures, and it explains them exactly: not approximately, not statistically, but as deductive consequences of the algebraic structure. The narrative origin story is, in this precise sense, the theory's empirical foundation.

Chapter 11

Manifesto

This chapter distils the preceding theory, experiments, and case studies into seven numbered principles. Each principle is grounded in a specific theorem or empirical result developed in the book; none are aspirational slogans. We follow the principles with a practitioner checklist—a concrete gate that any long-context system with endogenous pivot selection should pass before deployment—and close with an honest accounting of what this book does *not* claim.

11.1 Seven Principles

11.1.1 Principle 1: Validity Is Not Semantics

Raw validity—does the output satisfy the grammar, the schema, the constraint set?—is necessary but insufficient. When the output's meaning depends on an *endogenous pivot* selected by arg max over the output itself, a valid output with a substituted pivot is syntactically well-formed yet semantically different from the intended output. In our experiments, raw validity stays at 1.0 while pivot preservation drops to $\sim$0.35—a $\approx$ 65% silent substitution rate (Chapter 9). Any system that reports only raw validity is hiding potential semantic drift behind a perfect score.

Ground truth. The validity mirage experiments of Chapter 9 measure raw validity, pivot preservation, and semantic regret on the same corpora. The gap between the first metric and the second two is the empirical basis for this principle.

11.1.2 Principle 2: Endogenous Pivots Demand Pivot-Consistent Metrics

If your system selects a distinguished element from within the solution—a turning point, root cause, reference activity, schema anchor—you must measure whether that element is preserved under compression, truncation, and streaming. Standard metrics (accuracy, validity rate, BLEU, ROUGE) are oblivious to pivot identity; they evaluate the surface form without checking which element was chosen as the structural linchpin. The three diagnostics introduced in Chapter 9—*pivot preservation*, *fixed-pivot feasibility*, and *semantic regret*—are the minimum viable measurement framework for any system with endogenous coupling.

11.1.3 Principle 3: Greedy Has No Guarantees Under Endogenous Constraints

The feasible family under endogenous pivot selection violates both the hereditary axiom and the exchange axiom (Proposition 2.30). This structural violation means that greedy algorithms—the de-

fault workhorse for constrained selection in combinatorial optimisation—have *zero* approximation guarantees for endogenous pivot problems.

The absorbing-state theorem (Chapter 3) quantifies the worst case: when $j_{\text{dev}} < k$, greedy produces zero valid sequences. This is not a soft degradation; it is a hard structural impossibility. The event graph enters an absorbing state from which no continuation can reach acceptance, and the impossibility is detectable from the prefix alone.

11.1.4 Principle 4: Compression Must Be Algebra-Preserving

Compression is the unique closure-breaking operation in the context algebra: 13.3% of naive compressions violate monoid closure, while composition, pivot update, and split-at-point all have 0% violation rates (Chapter 7). Contract-guarded compression—preserving $d_{\text{total}} \geq \min(d_{\text{total}}, k)$—reduces silent mirages from 21.95% to 0% on real incident data (Chapter 7).

Treating compression as a free operation—"just drop some tokens"—is a safety failure. Every compression map must satisfy the no-absorption contract, or it risks pushing an algebraically healthy context element across the absorbing boundary into the ideal from which no continuation can produce a faithful output.

11.1.5 Principle 5: Commitment Timing Is a First-Class Design Dimension

Under commit-now streaming, 54.9% of organic sequences fall into oscillation traps (Chapter 8). Deferring commitment to $f \approx 0.25$ recovers 80.1% validity at 85% of offline quality. The choice of *when* to commit to a pivot is not an implementation detail—it is a fundamental design parameter with a $2\times$ impact on system reliability.

Every streaming system with endogenous pivot selection should explicitly specify and justify its commitment policy. The commitment fraction f interacts with the pivot arrival distribution; the experiments in Chapter 8 show that the optimal f varies by agent archetype, and that committing too early is strictly worse than committing too late.

11.1.6 Principle 6: Constraints Regularise; Relaxing Them Enables Exploitation

Strict grammar constraints achieve 88% validity; relaxed (strictly more permissive) constraints achieve 32% (Chapter 10). The constraints act as regularisers that prevent the optimisation metric from exploiting degenerate solutions. Removing constraints in the name of "flexibility" or "generality" opens the door to Goodhart collapse [10]: the metric is optimised, but the output is meaningless.

The single necessary constraint—minimum development beats ≥ 1—accounts for 100% of observed failures in the narrative extraction experiments of Chapter 10. This is a concrete instance of a general pattern: a small number of structural constraints do disproportionate regularisation work, and their removal has catastrophic (not graceful) consequences.

11.1.7 Principle 7: Measure and Report Mirage Gaps

For every system that performs constrained extraction, compression, or streaming with endogenous pivots, report the *mirage gap*:

$$\text{mirage gap} \;=\; \text{raw validity} - \text{pivot preservation}.$$

A mirage gap of 0 means the system is semantically faithful: every valid output preserves the intended pivot. A mirage gap > 0 means some fraction of "valid" outputs have silently substituted pivots. This number should appear in every evaluation table alongside accuracy and raw validity (Chapter 9).

The mirage gap is a single scalar that summarises the severity of the endogenous coupling problem for a given system configuration. It is cheap to compute (it requires only a pivot-identity check on outputs already produced) and impossible to game without actually preserving pivots.

11.2 Practitioner Checklist

Before deploying a long-context system with endogenous pivot selection, verify that every item below is satisfied. Each item maps to one or more of the seven principles and to a specific chapter of this book.

1. **Identify all endogenous pivots.** List every element in your system that is selected by arg max (or any selection operator) over the output itself. If no such element exists, the endogenous coupling problem does not apply. If one or more exist, every subsequent item is mandatory. (Chapter 2)

2. **Measure pivot preservation.** Under your compression, truncation, and streaming policy, what fraction of outputs preserve the pivot that would have been selected under the offline, uncompressed pipeline? Report this number. (Chapter 9)

3. **Measure fixed-pivot feasibility.** When forced to use the original (offline) pivot, can the system still produce a valid output? A low fixed-pivot feasibility rate means the system cannot even *express* the correct answer under its current constraints. (Chapter 9)

4. **Measure semantic regret.** What is the quality loss attributable to pivot substitution? Semantic regret isolates the cost of choosing a different pivot from the cost of other compression artefacts. (Chapter 9)

5. **Verify the no-absorption contract.** Your compression policy must satisfy d_{total}-preservation: d_{total} after compression must remain at least $\min(d_{\text{total}}, k)$. A single violation is sufficient to push a context element into the absorbing ideal. (Chapter 7)

6. **Specify an explicit commitment point.** Your streaming policy must name a commitment fraction f, justified by the pivot arrival distribution in your domain. "Commit immediately" ($f = 0$) is a valid choice only if you can demonstrate that the pivot arrival distribution is concentrated at the start of the sequence. (Chapter 8)

7. **Test constraints under relaxation.** Relax each grammar or schema constraint independently and measure the change in validity and pivot preservation. Constraints that cause a large validity drop when relaxed are acting as regularisers, not merely as validators. Do not remove them. (Chapter 10)

8. **Report the mirage gap.** Every evaluation table should include the mirage gap (raw validity minus pivot preservation) alongside raw validity. A mirage gap of zero is the target; a nonzero mirage gap is a quantified warning. (Chapter 9)

9. **Produce a deterministic witness.** Construct or identify at least one concrete instance—a specific input, a specific compression policy, a specific pivot—that demonstrates the mirage effect on your system. Understand its mechanism: which operation broke pivot preservation, and why. A system without a known witness has not been tested; it has merely been lucky. (Chapter 3, Chapter 9)

11.3 What This Book Does Not Claim

Honesty about scope is not a weakness; it is a prerequisite for the claims that remain. We record the following limitations explicitly.

- **The tropical semiring is not claimed to be unique or optimal.** We do not claim that the $(\max, +)$ tropical semiring (Chapter 6) is the only or the best algebraic framework for endogenous pivot problems. It is the one that naturally arises from the weight-comparison structure of arg max-based pivot selection. Other coupling structures (e.g., attention-based or learned pivots) may demand different semirings, and we regard the identification of such alternatives as open.

- **The commitment fraction $f = 0.25$ is not universally optimal.** The value $f \approx 0.25$ is the empirical optimum for the narrative-extraction domain studied in Chapter 8. It depends on the pivot arrival distribution, which varies by domain, agent archetype, and event-graph topology. We do not claim that 0.25 transfers to other settings without re-estimation.

- **Contract-guarded compression does not eliminate all failure modes.** The no-absorption contract (Chapter 7) eliminates the specific failure mode of silent pivot substitution caused by d_{total}-deficiency after compression. It does not address failure modes arising from weight perturbation, causal-graph corruption, or adversarial input construction. These are distinct problems that require distinct guarantees.

- **We have not conducted human evaluation studies.** All quality metrics reported in this book are automated: pivot preservation, fixed-pivot feasibility, semantic regret, and raw validity are computed programmatically from system outputs and ground-truth pivot identities. The gap between automated metrics and human judgement of narrative quality, causal correctness, or explanatory adequacy is an open empirical question.

- **The theory is developed on temporal DAGs with weight-based pivot selection.** Every theorem and experiment in this book operates on directed acyclic event graphs where the pivot is selected by arg max over a scalar weight function. Extension to other endogenous coupling structures—attention-based selection, learned pivot functions, multi-pivot systems, cyclic dependency graphs—is conjectured to exhibit analogous absorbing-state phenomena, but this conjecture is unproved. We flag it as the most important direction for future work (Chapter 12).

Chapter 12

Discussion and Future Work

This book began with a single collapsed arc—Diana's—and traced the failure through four layers of analysis: empirical observation, algebraic formalisation, streaming extension, and unified theory. This chapter steps back to survey the landscape. We first assemble the unified picture (Section 12.1), showing how the four constituent papers tell a single coherent story. We then catalogue research directions (Section 12.2), grading each by its current evidence level. Section 12.3 draws connections to systems beyond narrative generation. Section 12.4 offers brief closing reflections.

12.1 The Unified Picture

The four papers that make up this book's empirical and theoretical spine are not four independent contributions; they are four acts of a single argument. Each act addresses a different facet of the same structural phenomenon—the validity mirage—and each builds on the conclusions of its predecessors.

Paper 00: Narrative / Lorien [3] (Chapters 1–2). The endogenous pivot problem was discovered empirically through Diana's phase collapse. Across 50 seeds of a dinner-party simulation, Diana's arc failed in 9 seeds with a uniform signature: zero development beats, an extremely early turning point (median normalised position 0.13), and full candidate pools. The failure was not caused by data scarcity but by a structural interaction between protagonist-event injection and endogenous pivot selection. Grammar relaxation experiments confirmed that the standard grammar could not be satisfied when the pool's highest-weight events arrived too early. This empirical observation motivated every formal development that followed.

Paper 01: Absorbing States [4] (Chapters 3–4). The absorbing-state analysis formalised the impossibility. When $j_{\text{dev}} < k$—that is, when the number of development-eligible events before the argmax-selected pivot falls below the prefix requirement—greedy search produces zero valid sequences. The failure is not probabilistic; it is structural. Chapter 4 classified failures into four classes: Class A (absorbing state), Class B (pipeline coupling), Class C (commitment timing), and Class D (assembly compression) and constructed a hierarchy of solvers: greedy, enumerative, and TP-conditioned. Each level in the hierarchy trades computational cost for strictly greater coverage of the failure space.

Paper 02: Streaming [6] (Chapter 8). The streaming analysis extended the theory to online settings, where events arrive sequentially and labels must be emitted before the full sequence is known. The central result is that commit-now policies fall into oscillation traps—sequences in which the running-max pivot overtakes the committed pivot, leaving the policy in an absorbing state from which no continuation is grammar-valid. Empirically, such traps affect 54.9% of organically generated sequences. Two solutions emerged: deferred commitment (with patience parameter f) and tropical streaming, which maintains a weight vector over all possible pre-pivot development counts and defers role assignment until the vector stabilises.

Paper 03: Context Algebra [5] (Chapters 5, 6, 7, 9). The context algebra unified everything into a single algebraic framework. The context monoid $(\mathcal{C}, \otimes_{\text{endo}})$ compresses any contiguous block of events into a triple $(w^\star, d_{\text{total}}, d_{\text{pre}})$ that composes associatively. The extended monoid $(\bar{\mathcal{C}}, \otimes_{\text{commit}})$ adds a commitment flag κ and reveals that prefix-deficient elements form a left ideal—the absorbing ideal. Compression is the unique closure-breaking operation: it is the only elementary edit that can force a valid context into the absorbing ideal. The *validity mirage* is the quantitative gap between raw validity (which remains at 1.0 under compression) and pivot preservation (which drops to 0.35), confirming that standard metrics are structurally blind to the failure.

The unified thesis. Endogenous constraints—where a distinguished element is selected by argmax over the solution—create structural failure modes that standard validity metrics miss. The context algebra provides the formal framework to analyse, predict, and prevent these failures. The absorbing ideal explains *why* failures are irrecoverable; the tropical lift explains *when* they become detectable; the streaming analysis explains *how often* they occur in practice; and the compression contract explains *what operations* must be guarded to avoid them.

12.2 Research Directions

The following directions are ordered roughly by distance from the current results. Each is labelled with its evidence status.

12.2.1 Set-Valued Algebra / Context Semiring

STATUS: CONJECTURED.

The context monoid tracks a single best weight per slot. When pivot identity is uncertain—for example, when two focal events have nearly equal weight—a single-pivot summary discards information that may be decision-relevant. A natural extension is to replace the scalar $w^\star$ with a *set* of candidate pivots, each annotated with its own feasibility profile. The resulting structure would be a context *semiring* rather than a monoid: addition corresponds to maintaining the non-dominated frontier of pivot candidates, and multiplication corresponds to block composition.

The tropical context (Chapter 6) already moves in this direction. The weight vector $W[0..k]$ records the best pivot weight achievable with exactly j pre-pivot development events, for each j. A full set-valued semiring would generalise this to maintain the non-dominated Pareto frontier over an arbitrary set of quality dimensions, not just the development count.

The conjecture is that such a semiring admits an efficient parallel reduction analogous to the $O(\log n)$ holographic tree, with the set-valued composition reducing to a tropical matrix product. No proof or implementation exists at present.

12.2.2 Agent-Relative Tension Metrics

STATUS: PROPOSED.

Diana's failures occurred partly because the composite quality metric Q valued kinetic events—catastrophes, confrontations, revelations—over epistemic events—observations, realisations, belief updates. The Q-metric measures global tension: how much the event changes the state of the world. An agent-relative metric would instead measure how much the event changes the *focal agent's beliefs*, regardless of the event's global salience.

Concretely, define the *agent-relative tension* of event e for focal actor $a^\star$ as the KL divergence between $a^\star$'s belief state before and after e. Turning points would then be selected by argmax over agent-relative tension rather than over global kinetic weight. This would naturally favour events that are informationally significant to the focal agent, even if they are globally unremarkable.

No implementation or evaluation exists. The proposal is motivated by the observation that Diana's highest-weight events (by the global Q-metric) were protagonist-driven catastrophes that Diana merely witnessed, while the events most relevant to Diana's own arc were quieter observations that scored low on global tension.

12.2.3 Quality-Diversity Search via MAP-Elites

STATUS: PROPOSED.

The current solver hierarchy explores the space of narrative arcs by varying the turning-point candidate (enumerative solver) or by conditioning on turning-point position (TP-conditioned solver). Neither method systematically explores the full frontier of turning-point position versus quality.

MAP-Elites [17] (multi-dimensional archive of phenotypic elites) offers a principled alternative, drawing on the insight from novelty search [15] that exploring diverse solutions can outperform purely objective-driven optimisation. The behaviour space is discretised into bins indexed by normalised turning-point position. Each bin stores the highest-Q arc whose turning point falls in that bin. The archive is populated by mutation and crossover over event-selection vectors.

The expected benefit is diagnostic: the MAP-Elites archive would reveal whether metric misalignment (the tendency of Q to favour early turning points) is a *local* phenomenon—affecting only a narrow band of turning-point positions—or a *global* one that distorts the entire quality landscape. If the archive shows high-Q arcs at late turning-point positions, the problem is local and can be addressed by reweighting. If the archive is empty at late positions, the problem is structural and requires a metric redesign.

12.2.4 Adaptive Temporal Injection Thresholds

STATUS: PARTIALLY VALIDATED.

The temporal injection filter—which excludes protagonist events arriving before a normalised position threshold of 0.20—recovered 7 out of 9 of Diana's failed arcs in the original experiments. The threshold 0.20 was hand-tuned.

A more principled approach would derive the threshold from the pivot arrival CDF developed in Chapter 8. Specifically, let $F_\tau(t)$ denote the cumulative distribution of the running-max pivot's arrival position. The adaptive threshold is then

$$t^\star \;=\; \inf\bigl\{\, t : F_\tau(t) \geq 1 - \epsilon \,\bigr\},$$

where ϵ is a tolerance parameter controlling how much early pivot mass is acceptable. This ties the injection filter directly to the record-process analysis: the threshold adapts to the pool's statistical properties rather than being fixed across all agents.

Partial validation comes from the observation that 7/9 recoveries at 0.20 is consistent with the record-process prediction that early pivots concentrate in the first quintile. A full validation would require sweeping ϵ over a range of agent archetypes and measuring the recovery rate as a function of the adaptive threshold.

12.2.5 Human Evaluation Gap

STATUS: ACKNOWLEDGED LIMITATION.

All quality metrics in this work are automated. The composite Q-score aggregates tension variance, peak tension, trajectory shape, irony, significance, thematic coherence, and protagonist coverage. Each component is computed from the event graph without human input. No human evaluation studies have been conducted.

The correlation between automated Q-score and human narrative quality judgement is unknown. It is entirely possible that arcs judged high-quality by the Q-metric are perceived as contrived or incoherent by human readers, or conversely that arcs with low Q-scores are compelling. This is the single largest open question for practical deployment of the theory.

A human evaluation study would require: (i) a corpus of generated arcs spanning the full range of Q-scores and turning-point positions; (ii) human raters scoring each arc on narrative coherence, emotional engagement, and structural satisfaction; and (iii) a correlation analysis between human scores and automated metrics, broken down by pivot preservation status. The last point is critical: if human raters reliably distinguish pivot-preserved arcs from mirage arcs, the validity mirage has perceptual consequences. If they do not, the mirage is a formal property with no practical impact.

12.2.6 Extension to LLM Context Management

STATUS: CONJECTURED.

The context algebra's results on compression and commitment timing may apply directly to several problems in large language model (LLM) infrastructure.

KV-cache eviction. When a transformer's key–value cache exceeds its budget, the system must evict entries. The retention sweep experiments in Chapter 9 showed that naive eviction (removing low-attention entries) can silently shift the semantic pivot of the context window. The compression contract (Chapter 7) predicts that any eviction policy lacking a pivot-preservation guard will eventually produce mirage states. The connection to attention sink methods (e.g., StreamingLLM's [25] initial-token retention) is suggestive: the "sink" tokens may function as pivot anchors, and their retention may be a special case of the no-absorption contract.

RAG chunk selection. Retrieval-augmented generation [16] selects chunks from a document store to form the context for a query. The selected chunks collectively determine the answer's semantic pivot—the dominant theme or fact around which the answer is organised. Chunk selection is thus an endogenous pivot problem: the "best" chunk depends on the answer, which depends on which chunks are selected. The context algebra predicts that greedy chunk selection (top-k by embedding similarity) will produce mirage states when the highest-similarity chunk anchors the answer to a subtopic that lacks sufficient supporting detail.

Multi-document summarisation. Summarising multiple documents under a token budget requires compressing context. The dominant theme of the summary—its pivot—is determined by the compressed representation. The algebra predicts that naive compression (uniform truncation) will shift the pivot toward whichever document's key sentences survive truncation, regardless of whether that document is globally dominant. Contract-guarded compression would check, after each truncation step, whether the pivot has survived.

These connections are plausible but unproved beyond the KV-cache experiments reported in Chapter 9.

12.3 Connections to Broader Systems

The endogenous pivot problem is not specific to narrative generation. Any system that selects a distinguished element by optimising over its own output—including generative agent architectures [19]—faces the same structural coupling. This section sketches five domains where the theory applies and states what the algebra predicts.

12.3.1 LLM Context Management

Modern large language models operate under strict context-window budgets. When the budget is exceeded, the system must compress: KV-cache entries are evicted, attention windows are chunked, or prefixes are summarised. Each of these operations is a lossy context reduction.

Endogenous pivot analogue. The semantic pivot of the context window is the token (or span of tokens) that most strongly determines the model's next-token distribution—the "attention sink" in the terminology of StreamingLLM. This pivot is endogenous: it depends on the full context, yet compression modifies the context.

Theory predicts. The compression contract (Chapter 7) predicts that any eviction policy that does not explicitly check pivot preservation will produce mirage states: the model's output will remain fluent and locally coherent (high raw validity) while the semantic anchor of the generation has silently shifted (low pivot preservation). Attention sink methods that retain initial tokens may be understood as an implicit pivot-preservation heuristic—retaining the tokens most likely to serve as the global attention anchor.

12.3.2 Incident Triage and Root Cause Analysis

Incident investigation—whether in aviation (NTSB reports), software (post-incident reviews), or industrial safety—requires identifying a *root cause* from a sequence of events. The root cause is the event that, if removed, would have prevented the incident.

Endogenous pivot analogue. The root cause is selected by argmax over causal contribution, which depends on which events are included in the analysis. When the event log is truncated (e.g., limited to the final 60 minutes before the incident), the root cause may shift to a proximate trigger that happened to fall within the window, even though the true root cause occurred hours earlier.

Theory predicts. The absorbing-state theorem predicts that truncation-induced pivot shifts are irrecoverable: once the true root cause is outside the analysis window, no amount of reasoning over the truncated log can recover it. The tropical context predicts that maintaining a ranked list of candidate root causes (not just the top one) provides a natural defence: if the second-best candidate is qualitatively different from the best, the analysis is fragile and the window should be extended.

12.3.3 Process Mining

Process mining [23] discovers workflow models from event logs. A central task is *reference activity selection*: choosing the activity that anchors the alignment of all traces.

Endogenous pivot analogue. The reference activity is the process-mining analogue of the turning point. It is typically selected as the most frequent or most informative activity—an argmax over the discovered model. But the model depends on how traces are aligned, which depends on the reference activity.

Theory predicts. When event logs are sampled or filtered (e.g., retaining only the most recent n cases), the reference activity may shift to a different activity that happens to dominate the filtered sample. The context algebra predicts that this shift is absorbing under committed semantics: once downstream analyses are conditioned on the wrong reference activity, no subsequent correction can recover the original alignment without re-processing the full log.

12.3.4 Grammar-Constrained Decoding

Grammar-constrained decoding systems—such as PICARD [21] (for SQL), Outlines [24] (for structured JSON), and constrained beam search [7]—guide language model generation to satisfy a formal grammar.

Endogenous pivot analogue. In many structured outputs, a single token or clause serves as the semantic pivot: the `WHERE` clause in SQL, the root key in JSON, the topic sentence in a paragraph. This pivot is endogenous: it emerges from the generation process and determines the meaning of surrounding tokens.

Theory predicts. Grammar constraints ensure syntactic validity but cannot enforce semantic coherence with respect to the pivot. A grammar-valid SQL query may satisfy all syntactic rules while selecting the wrong `WHERE` predicate—a pivot shift that changes the query's meaning without triggering a grammar violation. The theory predicts that grammar constraints are necessary but not sufficient: they must be augmented with pivot-aware semantic checks. This is precisely the "grammar as regulariser, not validator" principle from Chapter 11.

12.3.5 Constrained Beam Search

Beam search with constraints maintains a set of B partial hypotheses, pruning at each step to the top-B candidates. When the output must satisfy a structural constraint (e.g., balanced parentheses, valid XML), the beam is further filtered by constraint satisfaction.

Endogenous pivot analogue. The semantic pivot of a partial hypothesis is the token or span that most strongly determines the hypothesis's eventual meaning. As the beam evolves, the pivot of the top hypothesis may oscillate—exactly the streaming oscillation trap of Chapter 8. When the beam commits to a hypothesis (by pruning all alternatives), the pivot is locked.

Theory predicts. The streaming analysis predicts that beam width B controls the effective patience parameter f: wider beams defer commitment longer and therefore have lower trap rates. The tropical context provides a natural beam-scoring function: instead of scoring hypotheses by log-likelihood alone, score them by their tropical feasibility vector, preferring hypotheses that maintain feasibility at multiple development thresholds. This is a concrete instantiation of deferred commitment in the beam-search setting.

12.4 Closing Remarks

The validity mirage is not a bug in a specific system. It is a structural property of any system where interpretation depends on endogenous selection—where the meaning of the output is determined by an element that is itself selected from the output. Such systems are ubiquitous: narrative generators select turning points, incident analysers select root causes, process miners select reference activities, retrieval systems select anchor documents, and language models select attention sinks. In every case, standard validity metrics ask "does the output satisfy the constraints?" and miss the deeper question: "does the output mean what it was supposed to mean?"

The algebra developed in this book provides tools to see the mirage (the pivot-preservation metric), measure it (the mirage gap between raw validity and fixed-pivot feasibility), and prevent it (the compression contract, the deferred commitment policy, the tropical streaming algorithm). The manifesto of Chapter 11 distils these tools into a practitioner's checklist. The research directions catalogued in Section 12.2 give researchers a roadmap.

The central message is simple. Validity is necessary but not sufficient. Compression is dangerous but manageable. Commitment is irreversible but deferrable. And the algebra is there to tell you which is which.

Appendix A

Notation

Chapter A collects every symbol used in this book, together with a brief gloss and the chapter in which it is introduced.

Symbol	Meaning	Introduced
$G = (V, E, t, w, a)$	Event graph: vertices V, edges E, timestamp function t, weight function w, actor function a	Chapter 2
$a^\star$	Focal actor: the agent whose narrative arc is being extracted	Chapter 2
P	Candidate pool: the set of events involving the focal actor	Chapter 2
$\tau(S)$	Endogenous turning point of event set S: the focal event with maximum weight in the selected subsequence	Chapter 2
$A = (Q, \Sigma, \delta, q_0, F)$	Phase grammar DFA: states Q, alphabet Σ (phase labels), transition function δ, initial state q_0, accepting states F	Chapter 2
k	Prefix requirement: the minimum number of DEVELOPMENT beats before the turning point	Chapter 2
M	Solver budget: the number of pivot candidates explored by the enumerative or TP-conditioned solver	Chapter 2
j_{dev}	Development-eligible event count: the number of non-focal events whose timestamp precedes the argmax-selected pivot	Chapter 3
$C = (w^\star, d_{\text{total}}, d_{\text{pre}})$	Context element: a triple summarising the structural state of a contiguous block of events	Chapter 5
$\otimes_{\text{endo}}$	Endogenous composition: the associative binary operation on context elements	Chapter 5
$e = (-\infty, 0, 0)$	Identity element of the context monoid	Chapter 5
$T = (W, d_{\text{total}})$	Tropical context: a weight vector W paired with a total development count	Chapter 6
$W[j]$	Best pivot weight achievable with exactly j pre-pivot development events	Chapter 6
$\bar{C} = (w^\star, d_{\text{total}}, d_{\text{pre}}, \kappa)$	Extended context element: a context element augmented with a commitment flag	Chapter 7
$\otimes_{\text{commit}}$	Committed composition: the composition operation on extended context elements that respects commitment	Chapter 7
κ	Commitment flag: $\kappa = 0$ (uncommitted, pivot is provisional) or $\kappa = 1$ (committed, pivot is locked)	Chapter 7
$\perp$	Absorbing predicate: $d_{\text{pre}} < k$, indicating that the context is in the absorbing ideal	Chapter 7
μ	Compression map: a lossy reduction of the context that may shift the pivot	Chapter 7
τ_t	Running-max pivot at step t: the highest-weight focal event seen in the first t events of the stream	Chapter 8
p_{eff}	Effective pre-pivot capacity: the number of development-eligible events before the running-max pivot in a streaming setting	Chapter 8
f	Patience parameter: the fraction of the sequence that the deferred commitment policy waits before locking the pivot	Chapter 8
min_gap	Minimum inter-record gap: the smallest number of events between consecutive running-max pivot updates	Chapter 8
PPR	Pivot preservation rate: the fraction of sequences in which the post-compression pivot matches the pre-compression pivot	Chapter 9
FPF	Fixed-pivot feasibility: the fraction of sequences that remain grammar-valid when evaluated against the original (pre-compression) pivot	Chapter 9
SR	Semantic regret: the quality loss attributable to pivot shift, measured as $Q_{\text{original}} - Q_{\text{shifted}}$	Chapter 9
Q	Composite quality score: a weighted aggregate of tension variance, peak tension, trajectory shape, irony, significance, thematic coherence, and protagonist coverage	Chapter 10
VA	Validity-adjusted score: a metric that interpolates between raw validity and pivot-preserving validity	Chapter 10
α	Interpolation parameter: controls the weight of pivot preservation in the validity-adjusted score	Chapter 10

Appendix B

Glossary

Absorbing state
A state of the product automaton from which no accepting state is reachable. Once entered, no continuation can produce a grammar-valid sequence.

Blast radius The number of downstream labels or decisions invalidated by a single pivot shift.

Candidate pool
The set of events involving the focal actor that are available for inclusion in the narrative arc.

Commit-now policy
A streaming policy that irrevocably assigns phase labels to each event as it arrives, based on the current running-max pivot.

Committed semantics
The interpretation regime under which the pivot identity is locked and cannot be revised by future events.

Compression contract
A pre/post condition pair that a compression map must satisfy to guarantee that the pivot is preserved and the context does not enter the absorbing ideal.

Context element
A triple $(w^\star, d_{\text{total}}, d_{\text{pre}})$ that summarises the structural state of a contiguous block of events: the best pivot weight, the total development capacity, and the pre-pivot development capacity.

Deferred commitment
A streaming policy that withholds label assignment for the first fraction f of the sequence, committing only after the pivot has stabilised.

Development-eligible
An event is development-eligible if it is a non-focal event whose timestamp precedes the current pivot. Such events can serve as DEVELOPMENT beats in the phase grammar.

Endogenous pivot
A distinguished element of the output that is selected by argmax over the output itself, creating a circular dependency between selection and interpretation.

Enumerative solver
A solver that iterates over the top-M pivot candidates by weight, attempting grammar-valid extraction for each, and returns the first success.

Event graph The directed graph $G = (V, E, t, w, a)$ encoding all events in the simulation, with timestamps, weights, and actor assignments.

Feasibility A sequence is feasible if it satisfies the phase grammar with at least k development beats before the turning point selected by the endogenous pivot function.

Fixed-pivot feasibility (FPF)
The fraction of compressed sequences that remain grammar-valid when evaluated against the pre-compression pivot.

Focal actor The agent $a^\star$ whose narrative arc is being extracted from the event graph.

Grammar-aware classifier
A classifier that uses the phase grammar's state to prune or reweight events during arc extraction, treating the grammar as a regulariser rather than a post-hoc validator.

Mirage gap The quantitative difference between raw validity (which may be 1.0) and pivot-preserving metrics such as FPF or PPR. A large mirage gap indicates that standard metrics are hiding semantic failures.

Monotonic DFA
A DFA whose state transitions are monotonically ordered: once a phase is exited, it cannot be re-entered.

Oscillation trap
A streaming state in which the running-max pivot has overtaken the committed pivot, leaving the policy in an absorbing state from which no grammar-valid continuation exists.

Patience parameter
The fraction $f \in [0, 1]$ of the sequence that the deferred commitment policy processes before locking the pivot.

Phase grammar
A deterministic finite automaton that specifies the legal ordering of narrative phases: SETUP, DEVELOPMENT, TURNING_POINT, RESOLUTION.

Pivot preservation (PPR)
The fraction of sequences in which the post-operation pivot (after compression, streaming, or other transformation) matches the pre-operation pivot.

Prefix requirement
The parameter k specifying the minimum number of DEVELOPMENT beats that must precede the turning point for a sequence to be grammar-valid.

Record process
The stochastic process formed by the running maximum of pivot weights in a stream. Each new record corresponds to a pivot update.

Semantic regret
The quality loss caused by a pivot shift: the difference in composite Q-score between the arc built around the original pivot and the arc built around the shifted pivot.

Tropical context
A pair (W, d_{total}) where W is a weight vector indexed by pre-pivot development count. The tropical composition rule is a shift-and-max operation derived from the tropical semiring.

Turning point The event selected by the endogenous pivot function—the single highest-weight focal event in the chosen subsequence.

Validity mirage The phenomenon in which standard validity metrics (grammar satisfaction, constraint checks) report perfect scores while the semantic content of the output has been silently corrupted by a pivot shift.

Weight vector The vector $W[0..k]$ in a tropical context, where $W[j]$ is the best pivot weight achievable with exactly j pre-pivot development events.

Appendix C

Determinism Contract

Every experimental result in this book is deterministically reproducible. This appendix documents the mechanisms that guarantee reproducibility.

C.1 Fixed Random Seeds

Every random generator in the codebase takes an explicit seed parameter. No experiment relies on a global random state or on the default seed. Each test function creates its own `numpy.random.Generator` instance initialised with a fixed seed, ensuring that the random state of one test cannot contaminate another.

```
rng = numpy.random.default_rng(seed=42)
```

The seed value is recorded in the test file and in the output CSV. Re-running the test with the same seed on the same platform produces bit-identical results.

C.2 Deterministic Tie-Breaking

When two events have equal weight, the system breaks ties using a three-key tuple:

$$(w,\ -t,\ \texttt{event\_id}),$$

where w is the event weight, t is the timestamp (negated so that earlier events win ties), and `event_id` is a unique integer identifier assigned at event creation. This ensures a strict total order on events, eliminating any dependence on hash randomisation or memory layout.

C.3 Stable Sorting

All sorting operations use a stable sort with an explicit key function. No code path relies on the default comparison of complex objects or on the stability guarantees of a particular sorting algorithm beyond the Python specification that `list.sort()` and `sorted()` are stable. Where numpy sorting is used, `kind='stable'` is specified explicitly.

C.4 Isolated Random State

Each test creates its own `numpy.random.Generator` with a fixed seed. Tests do not share random state, and the order in which tests are executed does not affect their outputs. This isolation is enforced by convention: no test reads from or writes to a global random state.

C.5 Platform Independence

All results reported in this book were produced and verified on the following platform:

- Python 3.14
- NumPy (current stable release)
- macOS (Darwin)

No platform-specific numerical libraries (e.g., MKL, CUDA) are used in the core experimental pipeline. Floating-point operations follow IEEE 754 double-precision semantics. No experiment depends on nondeterministic GPU operations.

Appendix D

Experiment-to-Artifact Map

Chapter D links every major experimental claim in this book to the test file that produces it, the CSV artifact that records the raw data, and the figure (if any) that visualises the result. All paths are relative to the repository root.

Claim	Test	Paper Exp	Artifact CSV	Figure
Tropical exactness (0 violations)	test_01	P01 Exp 1	results/raw/test_01_exactness.csv	test_01_exactness_heatmap.png
Associativity (0 violations)	test_02	P01 Exp 2	results/raw/test_02_associativity.csv	test_02_associativity_heatmap.png
Monoid subsumption	test_03	P01 Exp 3	results/raw/test_03_monoid_subsumption.csv	test_03_monoid_richness.png
Absorbing ideal (0 violations)	test_04	P01 Exp 4	results/raw/test_04_absorbing_ideal.csv	test_04_absorption_escape_rates.png
Holographic exactness	test_05	P01 Exp 5	results/raw/test_05_holographic_exactness.csv	—
Incremental consistency	test_06	P01 Exp 6	results/raw/test_06_incremental_consistency.csv	—
Scaling $O(n \log n)$	test_07	P01 Exp 7	results/raw/test_07_scaling.csv	test_07_scaling_loglog.png
Divergence (super-linear)	test_08	P01 Exp 8	results/raw/test_08_divergence_raw.csv	test_08_divergence_loglog.png
Record process	test_09	P02 Exp 1	results/raw/test_09_record_process_raw.csv	test_09_records_and_min_gap.png
Tropical shield	test_10	P02 Exp 2	results/raw/test_10_tropical_shield_raw.csv	test_10_tropical_shield_policy_compare.png
Compression mirage	test_11	P03 Exp 1	results/raw/test_11_mirage_raw.csv	test_11_validity_mirage.png
Deterministic witness	test_12	P03 Exp 2	results/raw/test_12_deterministic_witness.csv	test_12_deterministic_witness.png
Contract compression	test_13	P03 Exp 3	results/raw/test_13_contract_compression_raw.csv	test_13_contract_vs_naive.png
Margin correlation	test_14	P03 Exp 4	results/raw/test_14_margin_correlation_raw.csv	test_14_margin_quartiles.png
Adaptive compression	test_15	P03 Exp 5	results/raw/test_15_adaptive_compression_raw.csv	test_15_adaptive_vs_uniform.png
Organic traps (54.9%)	test_16	P02 Exp 3	results/raw/test_16_organic_traps.csv	test_16_trap_rate_heatmaps.png
Tropical streaming	test_17	P02 Exp 4	results/raw/test_17_tropical_streaming.csv	test_17_policy_validity_bar.png
Transition depth	test_18	P03 Exp 6	results/raw/test_18_transition_vector_raw.csv	test_18_transition_vector_depth.png
5-model blackbox	—	P03 Ext	release/results/blackbox_bf16_5model/*.csv	—
KV cache evic		P03 Ext	release/results/	

Bibliography

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024. URL https://arxiv.org/abs/2404.14219.

[2] Anders Bjorner and Gunter M. Ziegler. Introduction to greedoids. In Neil White, editor, *Matroid Applications*, volume 40 of *Encyclopedia of Mathematics and its Applications*, pages 284–357. Cambridge University Press, 1992. doi: 10.1017/CBO9780511662041.011.

[3] Jack Gaffney. Continuous control and structural regularization in multi-agent narrative simulation. 2026. Working paper.

[4] Jack Chaudier Gaffney. Absorbing states in greedy search: When endogenous constraints break sequential extraction, 2026. arXiv preprint, submitted concurrently.

[5] Jack Chaudier Gaffney. The validity mirage: Context algebra for endogenous semantics under memory compression, 2026. arXiv preprint, submitted concurrently.

[6] Jack Chaudier Gaffney. Streaming oscillation traps in endogenous-pivot sequential extraction, 2026. arXiv preprint, submitted concurrently.

[7] Graeme Gange and Peter J. Stuckey. The argmax constraint. In *Principles and Practice of Constraint Programming (CP 2020)*, volume 12333 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2020. doi: 10.1007/978-3-030-58475-7_19.

[8] Jacob Garbe, Max Kreminski, Ben Samuel, Noah Wardrip-Fruin, and Michael Mateas. Story sifting. In *Proceedings of the International Conference on the Foundations of Digital Games (FDG)*, 2019.

[9] Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024. URL https://arxiv.org/abs/2408.00118.

[10] Charles A. E. Goodhart. Problems of monetary management: The UK experience. *Monetary Theory and Practice*, pages 91–121, 1984.

[11] Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

[12] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, *Column Generation*, pages 33–65. Springer, 2005. doi: 10.1007/0-387-25486-2_2.

[13] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

[14] Bernhard Korte and Laszlo Lovasz. Mathematical structures underlying greedy algorithms. In *Fundamentals of Computation Theory (FCT'81)*, volume 117 of *Lecture Notes in Computer Science*, pages 205–209. Springer, 1981. doi: 10.1007/3-540-10854-8_22.

[15] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020. URL https://arxiv.org/abs/2005.11401.

[17] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[18] Valery B. Nevzorov. *Records: Mathematical Theory*, volume 194 of *Translations of Mathematical Monographs*. American Mathematical Society, 2001.

[19] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

[20] Alfréd Rényi. On the extreme values of observations. *Selected Papers of Alfréd Rényi*, 3:50–65, 1962.

[21] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EMNLP-MAIN.779.

[22] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. *Advances in Neural Information Processing Systems*, 35, 2022.

[23] Wil van der Aalst. *Process Mining: Data Science in Action*. Springer, 2 edition, 2016. doi: 10.1007/978-3-662-49851-4.

[24] Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023. URL https://arxiv.org/abs/2307.09702.

[25] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023. URL https://arxiv.org/abs/2309.17453.

[26] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

[27] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Chao Wang, Haibin Li, Zhaozhuo Lin, et al. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023. URL https://arxiv.org/abs/2306.14048.