

Type-Safe Process-Evidence Engineering: A Mathematical Framework for Object-Centric Conformance Checking

Sean Chatman

Department of Computer Science

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

2026

*Keywords: process mining, object-centric event logs, Petri nets, conformance checking,
evidence algebra, differential geometry, Rust type system.*

Abstract

Process mining bridges the gap between process models and actual organisational behaviour by analysing event logs produced by information systems. The object-centric turn in process mining — driven by the OCEL 2.0 standard and Object-Centric Petri Nets — demands a mathematical framework capable of reasoning simultaneously about multiple interacting object types, rather than the single case-perspective inherited from the XES standard.

This dissertation develops such a framework from first principles, unifying four lines of enquiry.

R1 — Object-Centric Evidence Algebra. We define the *evidence algebra* $\mathfrak{E}$ as a graded R -algebra equipped with a lifecycle filtration $\{\mathfrak{E}_k\}$ whose associated graded pieces encode the admission, witnessing, and refusal stages of process-artefact lifecycles. We prove that every valid lifecycle morphism extends uniquely to a filtered algebra morphism, and identify the refusal cokernel with the named-refusal enums of the `wasm4pm-compatible` Rust library.

R2 — Soundness–Completeness Duality. We prove that for free-choice workflow nets, perfect token-replay fitness is equivalent to perfect precision under the OCEL log measure μ_λ . This Soundness–Completeness Duality Theorem provides a rigorous foundation for the empirical observation that well-structured process models cannot be simultaneously over-fitting (low precision) and perfectly compliant (high fitness).

R3 — Object-Centric Petri Nets with Type Preservation. We formalise Object-Centric Petri Nets with a type-preservation theorem guaranteeing that every binding-enabled firing preserves declared object types. We encode this theorem as a Rust typestate pattern — a zero-cost compile-time proof that no type-unsafe transition can occur — and extend the framework to Declare constraint-based models with a full 22-template catalogue and object-centric scope semantics.

R4 — Differential Geometry of Process Quality. We equip the quality space $(0, 1)^4$ with the Fisher–Rao Riemannian metric, derive closed-form geodesic equations, prove constant negative sectional curvature $K = -\frac{1}{4}$, and show that the fitness–precision trade-off is a consequence of this negative curvature. Gradient flow on the Lagrangian discovery objective is shown to converge at rate $O(1/k)$.

The four contributions are unified by a common theme: process quality is not a runtime observable but a *mathematical invariant* that can be encoded in types, proved algebraically, and optimised geometrically. The `wasm4pm-compatible` Rust library — published on crates.io — serves as the concrete instantiation of the abstract framework, demonstrating that the gap between formal theory and type-safe implementation can be closed without sacrificing either rigour or performance.

Keywords: process mining, object-centric event logs, OCEL 2.0, Petri nets, soundness, conformance checking, evidence algebra, differential geometry, Fisher–Rao metric, Rust type system, typestate, const-generics.

Acknowledgements

The intellectual debt of this dissertation is owed, first and foremost, to Professor Wil M. P. van der Aalst, whose twenty-five years of foundational work in process mining provided both the technical vocabulary and the scientific ambition that animate every page that follows. His insistence that *models must be grounded in evidence* — not constructed from idealisations — is the deepest methodological commitment of this work.

I am grateful to the developers and contributors of the `pm4py` library [15], whose open-source implementation of process discovery and conformance-checking algorithms provided an invaluable reference throughout the theoretical development of Chapters 5 and 8.

The Rust community — and in particular the designers of the nightly const-generic and generic-associated-type features — deserve acknowledgement for building a language expressive enough to host the type-theoretic constructions of Chapter 9. The Curry–Howard correspondence is not merely a theoretical curiosity in Rust; it is a design principle.

I thank the members of the Process and Data Science (PADS) group at RWTH Aachen for stimulating discussions on object-centric process mining and for their hospitality during a visiting research period.

Finally, I thank Mark — a loyal Dachshund and steadfast companion throughout the long hours of this project — for his unwavering commitment to guarding the writing desk.

Sean Chatman

2026

Contents

Abstract	i
Acknowledgements	iii
1 Introduction	2
1.1 The Object-Centric Turn in Process Mining	2
1.2 Historical Context	3
1.3 Motivation and Problem Statement	3
1.4 Scope and Contributions	4
1.5 Notation Table	4
1.6 Structure of the Dissertation	5
2 Mathematical Foundations	7
2.1 Sets, Relations, and Functions	7
2.2 Multisets	7
2.3 Traces and the Free Monoid	8
2.4 Partial Orders, Lattices, and Fixed Points	8
2.5 Groups, Rings, and Modules	9
2.6 Measure Theory Prerequisites	10
3 Petri Nets and Workflow Nets	11
3.1 Petri Nets	11
3.2 Workflow Nets	12
3.3 Free-Choice Nets and Hack’s Theorem	13
3.4 Category of Workflow Nets	13
3.5 Complexity of Soundness	14
4 Object-Centric Event Logs	15
4.1 Motivation and Historical Context	15
4.2 OCEL 2.0 Formal Definition	15
4.3 Log σ -Algebra and Measure	16
4.4 Flattening and Radon–Nikodým	17

4.5	OCEL Validation Laws	17
5	Conformance Checking	18
5.1	Token Replay	18
5.2	Alignment-Based Conformance	19
5.3	Quality Dimensions as Lebesgue Integrals	19
5.4	KKT Optimality for Discovery	20
6	Evidence Algebra	21
6.1	Evidence States as a Poset	21
6.2	The Evidence Algebra	21
6.3	Filtration	22
6.4	Lifecycle Morphism Theorem	22
6.5	Refusal Cokernel	23
6.6	Hopf Algebra Structure	24
7	Soundness–Completeness Duality	25
7.1	Preliminary Lemmas	25
7.2	Soundness–Completeness Duality Theorem	26
7.3	Corollaries	27
8	Object-Centric Petri Nets	28
8.1	Definition and Structure	28
8.2	Object-Centric Markings and Firing	29
8.3	Type Preservation	29
8.4	Binding Existence Theorem	29
8.5	OCPN Soundness	30
8.6	Type Preservation as Computational Refusal Law	30
9	Type-Theoretic Implementation in Rust	31
9.1	Overview and Design Philosophy	31
9.2	Curry-Howard Correspondence	32
9.3	Const-Generic Quality Bounds	32
9.3.1	The Nightly Type-Law Machinery	32
9.3.2	Dimension-Specific Newtypes	34
9.4	Typestate and Soundness Lifecycle	35
9.4.1	Motivation	35
9.4.2	Marker Types and PhantomData	36
9.5	Sealed-Capability Pattern	36
9.5.1	Definition	36
9.5.2	Comparison to Dependent Types	38

9.6	Named Refusal Enums	38
9.7	Runtime Fitness Newtype	40
9.8	Tests as Existence Proofs	40
9.9	Chapter Summary	42
10	Declare: Constraint-Based Process Specification	43
10.1	Motivation and Historical Context	43
10.2	LTL Foundations	44
10.3	Declare Template Catalogue	44
10.4	Arity and Polarity Functions	46
10.5	Translation to Workflow Nets	46
10.6	Object-Centric Declare	47
10.7	Implementation in <code>wasm4pm-compat</code>	48
10.8	Complexity Analysis	48
10.9	Chapter Summary	48
11	Differential Geometry of Quality Metrics	51
11.1	Motivation	51
11.2	The Quality Manifold	51
11.3	Geodesic Equations	52
11.4	Gradient Flow and Discovery Convergence	53
11.5	Curvature and Quality Trade-offs	54
11.6	Information-Geometric Interpretation of Duality	55
11.7	Connection to Evidence Algebra	55
11.8	Chapter Summary	56
12	Conclusions and Future Work	57
12.1	Summary of Contributions	57
12.2	Synthesis: A Unified Theoretical Picture	58
12.3	Open Problems	58
12.3.1	OP1: Stochastic Extension of the Duality Theorem	59
12.3.2	OP2: PSPACE-Hard Discovery under Declare Constraints	59
12.3.3	OP3: Differential Graded Algebra Structure on $\mathfrak{C}$	60
12.3.4	OP4: Adjoint Functor for the Flattening Map	60
12.3.5	OP5: Polynomial-Time Binding Existence Algorithm for OCPNs	60
12.4	Broader Impact and Limitations	61
12.4.1	Broader Impact	61
12.4.2	Limitations	61
12.5	Concluding Remarks	61

List of Figures

12.1	Dependency structure of the four theoretical contributions. Arrows indicate mathematical dependencies (each node uses results from its predecessors).	59
------	---	----

List of Tables

1.1	Principal notation.	4
10.1	Declare template catalogue. $a, b \in \Sigma$; $\mathbf{G}, \mathbf{F}, \mathbf{X}, \mathbf{U}$ are standard LTL operators.	45

Chapter 1

Introduction

The goal of process mining is to turn event data into insight and action.

W. M. P. van der Aalst, *Process Mining* (2016)

1.1 The Object-Centric Turn in Process Mining

Process mining [1] is the discipline that sits at the intersection of data science and process management. Given an *event log* — a structured record of events produced by an information system — it seeks to discover, monitor, and improve real processes. Since the seminal α -algorithm of van der Aalst *et al.* [1], the field has matured into a rich ecosystem of discovery, conformance, and enhancement techniques.

Until recently, the dominant abstraction was the *case-centric* event log, codified in the XES standard: each event belongs to exactly one case, and a process model is a description of how a single case evolves from creation to completion. This abstraction, while convenient, introduces a systematic distortion. Real information systems — ERP platforms, order management systems, healthcare workflows — involve *multiple interacting objects*: an order, its items, the customer, the shipment, the invoice. A case-centric log that flattens these relationships onto a single case perspective creates artificial divergence: the same physical event appears multiple times under different case identifiers, inflating the apparent complexity of the process and degrading the quality of discovered models.

The *object-centric* paradigm, formalised in the OCEL 2.0 standard [3], abandons the single-case fiction. An event may be associated with any number of objects of any number of types; the process is not a sequence of states for a single token but a web of interactions among a collection of evolving objects. This shift creates new theoretical demands: discovery algorithms, conformance checkers, and quality metrics must be generalised from the single-case setting to the multi-object setting without loss

of mathematical rigour.

1.2 Historical Context

The Petri-net foundations of process mining trace back to the work of Carl Adam Petri [8] and the workflow-net specialisation of van der Aalst [1]. The soundness characterisation of workflow nets — completeness, absence of deadlock, absence of dead transitions — provided the first formal quality criterion for process models.

The α -algorithm, inductive miner, and heuristic miner represent the main families of discovery algorithms; their quality is measured by the four dimensions of fitness, precision, generalisation, and simplicity [5]. Alignment-based conformance checking [16] brought optimality guarantees to the conformance problem. The Declare language [4] introduced constraint-based, flexible process specification.

The object-centric extension is younger. Berti and van der Aalst [2] formalised Object-Centric Petri Nets (OCPNs) and developed object-centric conformance checking. The OCEL 2.0 standard [3] standardised the event-log format. The `pm4py` library [15] implemented the algorithms. What has not yet been provided is a *unified mathematical framework* that derives all of these results from common algebraic and geometric foundations.

1.3 Motivation and Problem Statement

The central motivation for this dissertation is the observation that the theoretical foundations of process mining — multisets, Petri nets, measure-theoretic event logs, quality metrics — are not ad hoc constructs but instances of deep mathematical structures: free commutative monoids, graded algebras, Riemannian manifolds. Making these structures explicit serves three purposes:

1. **Unification:** results that appear as separate theorems become corollaries of a common algebraic or geometric principle.
2. **Precision:** informal claims about quality trade-offs (e.g., “fitness and precision are in tension”) become precise theorems with quantitative statements.
3. **Implementation:** a mathematical framework stated in terms of types and morphisms maps directly to type-safe code in languages such as Rust, closing the gap between theory and practice.

Problem 1.1 (Main Research Problem). Develop a unified mathematical framework for object-centric process-evidence engineering that:

- (P1) defines the algebraic structure of evidence artefact lifecycles,
- (P2) proves the deepest quality invariant relating fitness and precision,
- (P3) extends WF-net soundness to multi-type object interactions, and
- (P4) characterises the geometry of the process quality space and its implications for discovery convergence.

1.4 Scope and Contributions

The four contributions R1–R4 (stated in the Abstract) address P1–P4 respectively. The boundary of the work is as follows.

In scope. Petri nets; workflow nets; soundness; free-choice nets; OCEL 2.0 event logs; token-replay and alignment-based conformance; Declare constraint-based models; graded algebras; Fisher–Rao geometry; Rust type-theoretic implementation.

Out of scope. Bayesian process models; hardware-level performance analysis; natural-language process extraction; organisational mining; social network analysis; distributed systems; simulation.

1.5 Notation Table

Table 1.1 summarises the principal notation used throughout the dissertation.

Table 1.1: Principal notation.

Symbol	Meaning	Chapter
Σ	Activity alphabet	2
Σ^*	Set of all finite traces over Σ	2
$\mathcal{B}(X)$	Set of all finite multisets over X	2
$N = (P, T, F, W)$	Petri net with places, transitions, arcs, weights	3
M	Marking (place $\rightarrow$ non-negative integers)	3
$[N)$	Set of reachable markings from M_0	3
$RS(N, M)$	Reachability set	3
$\mathcal{L}$	Object-centric event log (OCEL 2.0)	4
$\mathcal{O}$	Set of objects	4
$\mathcal{A}$	Set of activities	4
$\mathcal{Q}$	Set of object types	4

Table 1.1 (continued)

Symbol	Meaning	Chapter
$\text{fit}(N, L)$	Token-replay fitness	5
$\text{prec}(N, L)$	Behavioural precision	5
$\text{gen}(N, L)$	Generalisation	5
$\text{simpl}(N)$	Simplicity	5
μ_λ	Log measure on trace space	5
$\mathfrak{E}$	Evidence algebra	6
$\mathfrak{E}_k$	k -th filtration stage of evidence algebra	6
$\phi_T(\vec{a})$	LTL formula for Declare template T	10
$\mathcal{M}$	Quality manifold $(0, 1)^4$	11
g_{ij}	Fisher–Rao metric tensor on $\mathcal{M}$	11
K	Sectional curvature of $\mathcal{M}$	11

1.6 Structure of the Dissertation

Chapter 2 establishes mathematical prerequisites: sets, multisets, free monoids, lattices, the Knaster–Tarski fixed-point theorem, graded algebras, and measure theory.

Chapter 3 develops Petri-net theory through workflow nets, soundness, free-choice nets, Hack’s theorem, and the category **WFNet**.

Chapter 4 formalises OCEL 2.0 as an 11-tuple, constructs its σ -algebra, and proves the flattening–Radon–Nikodým theorem.

Chapter 5 defines token-replay fitness, alignment-based fitness, and the four quality dimensions as Lebesgue integrals, then states the joint measurability theorem.

Chapter 6 constructs $\mathfrak{E}$ and proves the lifecycle morphism theorem.

Chapter 7 proves the Soundness–Completeness Duality Theorem with two corollaries.

Chapter 8 formalises OCPNs, binding existence, and type preservation.

Chapter 9 demonstrates the Curry–Howard encoding in Rust: const-generic bounds, tystate patterns, sealed capabilities.

Chapter 10 treats Declare templates, LTL semantics, the 22-template catalogue, and object-centric scope.

Chapter 11 develops the differential geometry of $\mathcal{M}$: Fisher–Rao metric, geodesics, curvature, convergence.

Chapter 12 summarises contributions R1–R4 and poses five open problems.

Chapter 2

Mathematical Foundations

Mathematics is the art of giving the same name to different things.

Henri Poincaré

2.1 Sets, Relations, and Functions

We assume standard Zermelo–Fraenkel set theory with the Axiom of Choice (ZFC). Let $\mathbb{N} = \{0, 1, 2, \dots\}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ denote the natural numbers, integers, rationals, and reals respectively.

Definition 2.1 (Partial and Total Functions). A *partial function* $f : X \rightarrow Y$ is a relation $f \subseteq X \times Y$ such that for each $x \in X$ there is at most one $y \in Y$ with $(x, y) \in f$. A *total function* $f : X \rightarrow Y$ has exactly one such y for every x . We write $\text{dom}(f)$ for the domain of definition of f and $\text{ran}(f)$ for its range.

Definition 2.2 (Powerset and Finite Subsets). For a set X , $\mathcal{P}(X)$ denotes the powerset and $\mathcal{P}_{\text{fin}}(X)$ the collection of all finite subsets.

2.2 Multisets

Definition 2.3 (Multiset (Bag)). A *multiset* (or *bag*) over a set X is a function $m : X \rightarrow \mathbb{N}$, where $m(x)$ is the *multiplicity* of x in m . The set of all multisets over X is denoted $\mathcal{B}(X)$.

Definition 2.4 (Multiset Operations). Let $m, m' \in \mathcal{B}(X)$.

1. *Sum*: $(m + m')(x) = m(x) + m'(x)$.
2. *Scalar multiple*: $(k \cdot m)(x) = k \cdot m(x)$ for $k \in \mathbb{N}$.

3. *Submultiset*: $m \leq m'$ iff $m(x) \leq m'(x)$ for all $x \in X$.
4. *Difference*: $(m - m')(x) = \max(0, m(x) - m'(x))$ when $m' \leq m$.
5. *Support*: $\lfloor m \rfloor = \{x \in X \mid m(x) > 0\}$.
6. *Size*: $|m| = \sum_{x \in X} m(x)$.

The *empty multiset* $\emptyset$ satisfies $\emptyset(x) = 0$ for all x .

Proposition 2.5 (Free Commutative Monoid). *$(\mathcal{B}(X), +, \emptyset)$ is the free commutative monoid over X : it is a commutative monoid, and for every commutative monoid $(M, *, e)$ and function $\phi : X \rightarrow M$, there exists a unique monoid homomorphism $\hat{\phi} : \mathcal{B}(X) \rightarrow M$ extending ϕ .*

Proof. Define $\hat{\phi}(m) = \prod_{x \in \lfloor m \rfloor} \phi(x)^{m(x)}$, where the product is taken in $(M, *)$ and $\phi(x)^{m(x)}$ denotes the $m(x)$ -fold $*$ -product of $\phi(x)$ with itself. Since M is commutative the product is well-defined regardless of the enumeration order of $\lfloor m \rfloor$. To check uniqueness: any homomorphism ψ extending ϕ must satisfy $\psi(m) = \psi(\sum_x m(x) \cdot [x]) = \prod_x \psi([x])^{m(x)} = \prod_x \phi(x)^{m(x)} = \hat{\phi}(m)$, where $[x]$ denotes the singleton multiset concentrated at x . $\square$

2.3 Traces and the Free Monoid

Definition 2.6 (Free Monoid). The *free monoid* Σ^* over an alphabet Σ is the set of all finite sequences (words) over Σ , with concatenation $\cdot$ and the empty word ε . For $w = a_1 a_2 \cdots a_n \in \Sigma^*$, $|w| = n$ is the *length* and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ the set of non-empty words.

Definition 2.7 (Trace Projection). For $w \in \Sigma^*$ and $A \subseteq \Sigma$, the *projection* $w \upharpoonright_A \in A^*$ is defined inductively by: $\varepsilon \upharpoonright_A = \varepsilon$; $(aw) \upharpoonright_A = a(w \upharpoonright_A)$ if $a \in A$; $(aw) \upharpoonright_A = w \upharpoonright_A$ if $a \notin A$.

2.4 Partial Orders, Lattices, and Fixed Points

Definition 2.8 (Partial Order). A *partial order* $(P, \leq)$ is a set P equipped with a binary relation $\leq$ that is reflexive, antisymmetric, and transitive. A *total order* also satisfies: $\forall x, y \in P, x \leq y$ or $y \leq x$.

Definition 2.9 (Lattice). A *lattice* $(L, \leq)$ is a partial order in which every pair of elements $\{x, y\} \subseteq L$ has a least upper bound (join) $x \vee y$ and a greatest lower bound (meet) $x \wedge y$. A *complete lattice* has joins and meets for all subsets (not just pairs).

Definition 2.10 (Monotone Function). $f : L \rightarrow L$ is *monotone* (order-preserving) if $x \leq y \Rightarrow f(x) \leq f(y)$.

Theorem 2.11 (Knaster–Tarski Fixed-Point Theorem). *Let $(L, \leq)$ be a complete lattice and $f : L \rightarrow L$ monotone. Then:*

1. *The set $\text{Fix}(f) = \{x \in L \mid f(x) = x\}$ is non-empty.*
2. *The least fixed point $\mu f = \bigwedge \{x \in L \mid f(x) \leq x\}$ and greatest fixed point $\nu f = \bigvee \{x \in L \mid x \leq f(x)\}$ both exist and are elements of $\text{Fix}(f)$.*

Proof. Existence of μf . Let $D = \{x \in L \mid f(x) \leq x\}$ (the set of *pre-fixed-points*). Since L is a complete lattice, $d = \bigwedge D$ exists. For any $x \in D$, $d \leq x$, so by monotonicity $f(d) \leq f(x) \leq x$. Since this holds for all $x \in D$, we have $f(d) \leq \bigwedge D = d$, i.e., $f(d) \leq d$, so $d \in D$. Applying monotonicity once more: $f(f(d)) \leq f(d)$, so $f(d) \in D$. By minimality of d : $d \leq f(d)$. Combined with $f(d) \leq d$, we get $f(d) = d$. Therefore $d \in \text{Fix}(f)$ and d is minimal among D (which contains all fixed points by definition), so $d = \mu f$.

Existence of νf . Symmetric: take $\bigvee \{x \mid x \leq f(x)\}$. □

Corollary 2.12 (Reachability as Least Fixed Point). *Let $N = (P, T, F, W)$ be a Petri net with initial marking M_0 . Define $\mathcal{F} : \mathcal{P}(\mathbb{N}^P) \rightarrow \mathcal{P}(\mathbb{N}^P)$ by*

$$\mathcal{F}(\mathcal{S}) = \{M_0\} \cup \{M' \mid \exists M \in \mathcal{S}, \exists t \in T, M[t]M'\}.$$

The reachability set $RS(N, M_0) = \mu \mathcal{F}$ is the least fixed point of the monotone function $\mathcal{F}$ on the complete lattice $(\mathcal{P}(\mathbb{N}^P), \subseteq)$.

2.5 Groups, Rings, and Modules

Definition 2.13 (Ring). A *ring* $(R, +, \cdot)$ is a set R with two binary operations satisfying: $(R, +)$ is an abelian group with identity 0; $(R, \cdot)$ is a monoid with identity 1; multiplication distributes over addition. A *commutative ring* has $a \cdot b = b \cdot a$ for all a, b .

Definition 2.14 (Graded Algebra). Let R be a commutative ring. An *R -algebra* A is a ring A together with a ring homomorphism $\phi : R \rightarrow Z(A)$ (the centre of A). A is *graded* (by $\mathbb{N}$) if $A = \bigoplus_{k=0}^{\infty} A_k$ as R -modules, with $A_i \cdot A_j \subseteq A_{i+j}$ for all i, j . The elements of A_k are called *homogeneous of degree k* .

Definition 2.15 (Filtered Algebra). An *R -algebra filtration* of A is a sequence of R -submodules $A = F_0 \supseteq F_1 \supseteq F_2 \supseteq \dots$ with $F_i \cdot F_j \subseteq F_{i+j}$. The *associated graded algebra* is $\text{gr}(A) = \bigoplus_{k \geq 0} F_k / F_{k+1}$.

2.6 Measure Theory Prerequisites

Definition 2.16 (σ -Algebra and Measure Space). A σ -algebra over a set X is a collection $\mathcal{F} \subseteq \mathcal{P}(X)$ containing $\emptyset$, closed under complementation and countable union. A *measure space* is a triple $(X, \mathcal{F}, \mu)$ where $\mu : \mathcal{F} \rightarrow [0, \infty]$ is σ -additive with $\mu(\emptyset) = 0$.

Theorem 2.17 (Carathéodory Extension). *Let $\mathcal{A}$ be an algebra on X and $\mu_0 : \mathcal{A} \rightarrow [0, \infty]$ a σ -finite pre-measure. Then there exists a unique measure μ on the σ -algebra $\sigma(\mathcal{A})$ generated by $\mathcal{A}$ that extends μ_0 .*

Theorem 2.18 (Radon–Nikodým). *Let $(X, \mathcal{F}, \mu)$ be a σ -finite measure space and $\nu \ll \mu$ (i.e., ν is absolutely continuous with respect to μ). Then there exists a μ -integrable function $f : X \rightarrow [0, \infty)$, the Radon–Nikodým derivative $\frac{d\nu}{d\mu}$, such that $\nu(A) = \int_A f d\mu$ for all $A \in \mathcal{F}$.*

These two theorems underpin the measure-theoretic treatment of event logs in Chapter 4 and the integral quality dimensions in Chapter 5.

Chapter 3

Petri Nets and Workflow Nets

Workflow nets provide the formal basis for verifying process models before they are deployed in practice.

W. M. P. van der Aalst, *Verification of Workflow Nets* (1997)

3.1 Petri Nets

Definition 3.1 (Petri Net). A *Petri net* is a quadruple $N = (P, T, F, W)$ where:

- P is a finite set of *places*;
- T is a finite set of *transitions*, with $P \cap T = \emptyset$;
- $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation* (arc set);
- $W : F \rightarrow \mathbb{N}^+$ is the *arc weight function*.

A *marking* is a function $M : P \rightarrow \mathbb{N}$ (token distribution). A *marked Petri net* is a pair (N, M_0) with M_0 the initial marking.

Definition 3.2 (Pre- and Post-Sets). For $x \in P \cup T$, the *preset* is $\bullet x = \{y \mid (y, x) \in F\}$ and the *postset* is $x^\bullet = \{y \mid (x, y) \in F\}$.

Definition 3.3 (Incidence Matrix). The *incidence matrix* of N is the integer matrix $\mathbf{C} \in \mathbb{Z}^{|P| \times |T|}$ with $\mathbf{C}[p, t] = W(t, p) - W(p, t)$, where $W(x, y) = 0$ if $(x, y) \notin F$.

Definition 3.4 (Enabling and Firing). Transition $t \in T$ is *enabled* at marking M , written $M[t]$, if $M(p) \geq W(p, t)$ for all $p \in \bullet t$. Firing t at M produces the successor marking $M' = M + \mathbf{C}[\cdot, t]$, written $M[t]M'$.

Definition 3.5 (State Equation). For a firing sequence $\sigma = t_1 t_2 \cdots t_n$ from M_0 , the final marking satisfies the *state equation*:

$$M_n = M_0 + \mathbf{C} \cdot \vec{\sigma},$$

where $\vec{\sigma} \in \mathbb{N}^{|T|}$ is the *Parikh vector* of σ ($\vec{\sigma}[t]$ = number of times t fires in σ).

Definition 3.6 (P- and T-Invariants). A *P-invariant* is a vector $\mathbf{y} \in \mathbb{Z}^{|P|}$ with $\mathbf{y}^\top \mathbf{C} = \mathbf{0}$. A *T-invariant* is a vector $\mathbf{x} \in \mathbb{N}^{|T|}$ with $\mathbf{C}\mathbf{x} = \mathbf{0}$.

3.2 Workflow Nets

Definition 3.7 (Workflow Net). A *workflow net* (WF-net) is a Petri net $N = (P, T, F, W)$ satisfying:

1. There is a unique *source place* $i \in P$ with $\bullet i = \emptyset$.
2. There is a unique *sink place* $o \in P$ with $o^\bullet = \emptyset$.
3. Every node $x \in P \cup T$ lies on a directed path from i to o .

The *short-circuit net* $\overline{N}$ is obtained from N by adding an arc (o, i) (with weight 1).

Definition 3.8 (Soundness). A WF-net N with initial marking $[i]$ (one token in i) is *sound* if:

1. **(S1) Option to complete:** for every reachable marking $M \in RS(N, [i])$, there exists a firing sequence to $[o]$.
2. **(S2) Proper completion:** if $M \geq [o]$, then $M = [o]$.
3. **(S3) No dead transitions:** for every $t \in T$ there exists $M \in RS(N, [i])$ with $M[t] \neq \emptyset$.

Theorem 3.9 (Short-Circuit Soundness Characterisation). *A WF-net N is sound if and only if the short-circuit net $\overline{N}$ is live and bounded.*

Proof. ($\Rightarrow$) Assume N is sound. Let M be any reachable marking of $\overline{N}$ from $[i]$. We show liveness (every transition is enabled from some reachable marking) and boundedness (markings are bounded).

Liveness. Any marking M reachable in $\overline{N}$ corresponds to a marking $M|_P$ in N . By (S1), there is a sequence in N from $M|_P$ to $[o]$. Firing the added arc (o, i) returns a token to i , restarting the cycle. By (S3), every transition is reachable from $[i]$ in $\overline{N}$.

Boundedness. By (S2), whenever $[o]$ is reached the token count in P is exactly 1. Since the net is finite and all reachable markings are finite, the net is bounded.

($\Leftarrow$) Suppose $\overline{N}$ is live and bounded.

(S1). Given $M \in RS(N, [i])$, since $\overline{N}$ is live the transition corresponding to the added arc (o, i) is reachable from M , which requires $[o]$ to be reachable.

(S2). If $M \geq [o]$, the extra token causes the added arc to fire, but since $\overline{N}$ is 1-bounded (by (S2) applied inductively) this is impossible.

(S3). Liveness of $\overline{N}$ implies every $t \in T$ is reachable. $\square$

3.3 Free-Choice Nets and Hack's Theorem

Definition 3.10 (Free-Choice Net). A Petri net is *free-choice* if for every arc $(p, t) \in F$:

$$p^\bullet = \{t\} \text{ or } {}^\bullet t = \{p\}.$$

Equivalently: if two transitions share a common input place, they share *all* input places.

Theorem 3.11 (Hack's Theorem). *A free-choice WF-net N is sound if and only if it is well-handled: every S-component (maximal strongly connected sub-net containing exactly one token in $[\overline{N}]$) covers N .*

Proof sketch. This is Theorem 8.6 in [8]. The key insight is that in free-choice nets, liveness is equivalent to every S-component being visited at least once per cycle. Well-handledness is the graph-theoretic certificate of this condition. $\square$

3.4 Category of Workflow Nets

Definition 3.12 (Category **WFNet**). The category **WFNet** has:

- *Objects*: sound WF-nets.
- *Morphisms*: net morphisms $f : N \rightarrow N'$ — pairs (f_P, f_T) preserving incidence, weights, source, and sink.
- *Composition*: component-wise.

Theorem 3.13 (Soundness is Functorial). *The forgetful functor $U : \mathbf{WFNet} \rightarrow \mathbf{PetriNet}$ (forgetting soundness) reflects soundness: if N' is sound and $f : N \rightarrow N'$ is an injective morphism, then N is sound.*

Proof. An injective net morphism f embeds N as a sub-net of N' . By Theorem 3.9, soundness of N' implies $\overline{N'}$ is live and bounded. The sub-net $\overline{N}$ inherits liveness (every transition in N has a firable image in N') and boundedness (markings of N are dominated by those of N'). Applying Theorem 3.9 in reverse gives soundness of N . $\square$

3.5 Complexity of Soundness

Theorem 3.14 (PSPACE-Completeness). *Deciding soundness of a WF-net is PSPACE-complete.*

Proof sketch. PSPACE membership: soundness reduces to reachability in bounded Petri nets, which is in PSPACE. PSPACE-hardness: by reduction from the satisfiability of quantified Boolean formulas (QBF) [8]. $\square$

Remark 3.15. For free-choice nets, soundness can be decided in polynomial time via Hack's Theorem (Theorem 3.11), exploiting the structural constraints of the free-choice property.

Chapter 4

Object-Centric Event Logs

Traditional process mining assumes one case notion. Reality has many.

W. M. P. van der Aalst, *Object-Centric Process Mining* (2019)

4.1 Motivation and Historical Context

The XES standard assumes a single case identifier per event. Real information systems produce events related to multiple objects simultaneously: an order event touches the order, the items, the customer, and the carrier. Forcing such events onto a single case projection introduces *convergence* (one event appearing once per related object, duplicating it) and *divergence* (artificially splitting one interaction into multiple case-level events). Both pathologies degrade process discovery quality [2].

The OCEL 2.0 standard [3] corrects this by allowing each event to relate to any subset of objects.

4.2 OCEL 2.0 Formal Definition

Definition 4.1 (OCEL 2.0 Log). An *OCEL 2.0 log* is an 11-tuple

$$\mathcal{L} = (\mathcal{E}, \mathcal{A}, \mathcal{O}, \mathcal{Q}, \mathcal{A}_E, \mathcal{A}_O, \text{act}, \text{type}, \text{time}, \text{attr}_E, \text{attr}_O, \text{rel})$$

where:

1. $\mathcal{E}$ is a finite set of *event identifiers*;
2. $\mathcal{A}$ is a finite set of *activities*;
3. $\mathcal{O}$ is a finite set of *object identifiers*;

4. $\mathcal{Q}$ is a finite set of *object types*;
5. $\mathcal{A}_E, \mathcal{A}_O$ are finite sets of *event* and *object attribute names*;
6. $\text{act} : \mathcal{E} \rightarrow \mathcal{A}$ maps each event to its activity;
7. $\text{type} : \mathcal{O} \rightarrow \mathcal{Q}$ maps each object to its type;
8. $\text{time} : \mathcal{E} \rightarrow \mathbb{R}^+$ assigns a timestamp (seconds since epoch) to each event;
9. $\text{attr}_E : \mathcal{E} \times \mathcal{A}_E \rightarrow \mathcal{V}$ gives event attribute values ($\mathcal{V}$ is a value domain);
10. $\text{attr}_O : \mathcal{O} \times \mathcal{A}_O \rightarrow \mathcal{V}$ gives object attribute values;
11. $\text{rel} \subseteq \mathcal{E} \times \mathcal{O} \times \mathcal{Q}$ is the *event-object relation* ($\mathcal{Q} \subseteq \mathcal{A}_E$ is the set of *qualifiers*).

Definition 4.2 (Object Lifecycle Trace). For an object $o \in \mathcal{O}$, the *lifecycle trace* of o is the subsequence of events related to o , ordered by timestamp:

$$\tau(o) = e_1 e_2 \cdots e_k \text{ where } \{(e_i, o, *) \in \text{rel}\} = \{e_1, \dots, e_k\} \text{ and } \text{time}(e_1) \leq \cdots \leq \text{time}(e_k).$$

Definition 4.3 (Type Projection). For a type $q \in \mathcal{Q}$, the *type projection* $\mathcal{L}_q$ is the sub-log restricted to objects of type q :

$$\mathcal{L}_q = (\mathcal{E}_q, \mathcal{A}, \mathcal{O}_q, \{q\}, \dots) \text{ where } \mathcal{O}_q = \{o \in \mathcal{O} \mid \text{type}(o) = q\}.$$

4.3 Log σ -Algebra and Measure

Definition 4.4 (Trace Space σ -Algebra). Let $\Sigma = \mathcal{A}$. The *trace space* is $(\Sigma^*, \mathcal{F})$ where $\mathcal{F}$ is the σ -algebra generated by the *cylinder sets*:

$$C(w) = \{\sigma \in \Sigma^* \mid w \text{ is a prefix of } \sigma\}, \quad w \in \Sigma^*.$$

By Carathéodory's extension theorem (Theorem 2.17), any pre-measure μ_0 defined consistently on the algebra of cylinder sets extends uniquely to $\mathcal{F}$.

Definition 4.5 (Log Measure). Given a log $L = \{\sigma_1, \dots, \sigma_N\}$, the *empirical log measure* μ_λ is the discrete probability measure

$$\mu_\lambda(A) = \frac{|\{i \mid \sigma_i \in A\}|}{N} \quad \text{for } A \in \mathcal{F}.$$

4.4 Flattening and Radon–Nikodým

Proposition 4.6 (Flattening Map). *There is a measurable surjection $\pi_* : (\mathcal{L}_{\text{OCEL}}, \mathcal{F}_{\text{OCEL}}) \rightarrow (\mathcal{L}_{\text{XES}}, \mathcal{F}_{\text{XES}})$ mapping each OCEL log to its XES flattening (for a fixed chosen object type q). For any measure μ on $\mathcal{L}_{\text{OCEL}}$, the pushforward $(\pi_*)_*\mu = \mu \circ \pi_*^{-1}$ is a measure on $\mathcal{L}_{\text{XES}}$.*

Theorem 4.7 (OCEL–XES Radon–Nikodým). *Let μ_{OCEL} be a σ -finite measure on $\mathcal{L}_{\text{OCEL}}$ and $\mu_{\text{XES}} = (\pi_*)_*\mu_{\text{OCEL}}$ its pushforward. For any measurable $f : \mathcal{L}_{\text{XES}} \rightarrow [0, \infty)$:*

$$\int_{\mathcal{L}_{\text{XES}}} f d\mu_{\text{XES}} = \int_{\mathcal{L}_{\text{OCEL}}} (f \circ \pi_*) d\mu_{\text{OCEL}}.$$

In particular, if $\nu \ll \mu_{\text{OCEL}}$ then $(\pi_)_*\nu \ll \mu_{\text{XES}}$ and $\frac{d(\pi_*)_*\nu}{d\mu_{\text{XES}}} = \mathbb{E}_{\mu_{\text{OCEL}}} \left[\frac{d\nu}{d\mu_{\text{OCEL}}} \middle| \pi_*^{-1}(\mathcal{F}_{\text{XES}}) \right]$.*

Proof. The first identity is the change-of-variables formula for pushforward measures (standard measure theory). The Radon–Nikodým density formula follows from the tower property of conditional expectations: the density of ν with respect to μ_{OCEL} , when integrated over the fibres of π_* , gives the density of $(\pi_*)_*\nu$ with respect to μ_{XES} . $\square$

4.5 OCEL Validation Laws

Definition 4.8 (OCEL Validation Laws). An OCEL log $\mathcal{L}$ is *well-formed* if it satisfies the following laws:

(L1) *Non-empty relation:* $\text{rel} \neq \emptyset$.

(L2) *Referential integrity:* for every $(e, o, q) \in \text{rel}$, $e \in \mathcal{E}$ and $o \in \mathcal{O}$.

(L3) *Temporal consistency:* for every $o \in \mathcal{O}$ and consecutive events e_i, e_{i+1} in $\tau(o)$, $\text{time}(e_i) \leq \text{time}(e_{i+1})$.

(L4) *Type closure:* for every $(e, o, q) \in \text{rel}$, $\text{type}(o) \in \mathcal{Q}$.

Remark 4.9. Laws (L1)–(L4) correspond directly to the named refusal variants of the `wasm4pm-compat` Rust library: $L_1 \leftrightarrow \text{EmptyEventObjectLinks}$; $L_2 \leftrightarrow \text{DanglingEventObjectLink}$. Laws L_3 and L_4 are enforced by the `Trace::validate()` method in the `eventlog` module.

Chapter 5

Conformance Checking

Conformance checking is about measuring the degree to which a process model and the reality captured in the event log agree.

W. M. P. van der Aalst, *Process Mining* (2016)

5.1 Token Replay

Definition 5.1 (Token Replay). Given a WF-net $N = (P, T, F, W)$ and a trace $\sigma = a_1 \cdots a_n \in \mathcal{A}^*$, *token replay* simulates σ on N starting from $[i]$ by:

1. For each activity a_k , find a transition t_k with label a_k that is enabled; fire t_k to advance the marking.
2. If no such t_k is enabled, *force-fire* by adding missing tokens to $\bullet t_k$ (recording a *missing-token count* m_k) and consuming tokens from $\bullet t_k$ that have no corresponding event (recording a *remaining-token count* r_k).
3. At the end, record the number of remaining tokens r_{final} not in $[o]$.

Let $p = \sum_k (m_k + r_k) + r_{\text{final}}$ be the total *misfit count* and $q = \sum_k (m_k^{\text{produced}} + r_k^{\text{consumed}})$ the total *token count*.

Definition 5.2 (Token-Replay Fitness). The *token-replay fitness* of a trace σ on N is

$$\text{fit}_{\text{tr}}(\sigma, N) = \frac{1}{2} \left(1 - \frac{m}{p} \right) + \frac{1}{2} \left(1 - \frac{r}{c} \right),$$

where m is the number of missing tokens, r the number of remaining tokens, p the number of produced tokens, and c the number of consumed tokens. The *log-level token-replay fitness* is

$$\text{fit}(N, L) = \int_{\Sigma^*} \text{fit}_{\text{tr}}(\sigma, N) d\mu_\lambda(\sigma).$$

5.2 Alignment-Based Conformance

Definition 5.3 (Alignment). An *alignment* of trace $\sigma \in \mathcal{A}^*$ against N is a sequence $\gamma = (a_1, b_1)(a_2, b_2) \cdots (a_k, b_k)$ where:

- Each $a_j \in \mathcal{A} \cup \{\gg\}$ (*log moves*);
- Each $b_j \in \mathcal{A} \cup \{\gg\}$ (*model moves*);
- $a_j = \gg$ indicates a *model move on model* (invisible in log);
- $b_j = \gg$ indicates a *log move on log* (not in model);
- Ignoring $\gg$ symbols: the sequence of a_j 's yields σ and the sequence of b_j 's is a valid firing sequence of N .

Definition 5.4 (Alignment Cost and Optimal Alignment). The *cost function* $\delta : (\mathcal{A} \cup \{\gg\})^2 \rightarrow \mathbb{R}_{\geq 0}$ assigns zero to synchronous moves (a, a) , and positive cost to log moves $(a, \gg)$ and model moves $(\gg, b)$. The *optimal alignment* is

$$\gamma^*(\sigma, N) = \arg \min_{\gamma \in \Gamma(\sigma, N)} \sum_j \delta(a_j, b_j),$$

where $\Gamma(\sigma, N)$ is the set of all alignments of σ against N .

Remark 5.5. Optimal alignments are computed using the A* shortest-path algorithm over the *synchronous product net* of N and the trace automaton of σ [18]. The state space is $|P| \times |\sigma|$, giving time complexity $O(|P| \cdot |\sigma| \cdot \log(|P| \cdot |\sigma|))$ with appropriate heuristics.

5.3 Quality Dimensions as Lebesgue Integrals

Definition 5.6 (Four Quality Dimensions). Let N be a WF-net and L an event log with log measure μ_λ .

$$\text{fit}(N, L) = \int_{\Sigma^*} \frac{|\text{synchronous moves in } \gamma^*(\sigma, N)|}{|\sigma| + |\text{model moves in } \gamma^*(\sigma, N)|} d\mu_\lambda(\sigma), \quad (5.1)$$

$$\text{prec}(N, L) = \int_{\Sigma^*} \frac{|en(M_\sigma) \cap \text{Activities}(L)|}{|en(M_\sigma)|} d\mu_\lambda(\sigma), \quad (5.2)$$

$$\text{gen}(N, L) = 1 - \int_{\Sigma^*} \sum_{t \in T} \frac{\mathbf{1}_{t \text{ fires in } \sigma}}{|en_N|^2} d\mu_\lambda(\sigma), \quad (5.3)$$

$$\text{simpl}(N) = \frac{2}{|P| + |T| + 1}. \quad (5.4)$$

Here M_σ is the marking reached after replaying σ ; $en(M)$ is the set of enabled transitions at marking M ; and $|en_N|$ is the average number of enabled transitions across all reachable markings.

Theorem 5.7 (Joint Measurability). *For any WF-net N with finite reachability set $RS(N, M_0)$:*

1. *The function $\sigma \mapsto \text{fit}_{\text{tr}}(\sigma, N)$ is $(\mathcal{F}, \mathcal{B}([0, 1]))$ -measurable.*
2. *The function $\sigma \mapsto \text{prec}_{\text{tr}}(\sigma, N)$ is $(\mathcal{F}, \mathcal{B}([0, 1]))$ -measurable.*
3. *The joint function $\sigma \mapsto (\text{fit}_{\text{tr}}(\sigma, N), \text{prec}_{\text{tr}}(\sigma, N))$ is measurable as a function into $[0, 1]^2$.*

Proof. For (1): the token-replay cost $m(\sigma, N)$ and token counts p, c, r are non-negative integer-valued functions of σ . Each step of token replay determines a transition firing by a measurable selection (Kuratowski–Ryll–Nardzewski measurable selection theorem), and finite compositions of measurable functions are measurable. The resulting fitness formula is a ratio of measurable functions and hence measurable.

For (2): the set $en(M_\sigma)$ depends measurably on σ through the transition function of token replay; the cardinality $|en(M_\sigma) \cap \text{Activities}(L)|$ is a composition of a measurable function with a discrete function over a finite set.

For (3): the joint measurability follows from (1) and (2) by the product σ -algebra on $[0, 1]^2$. $\square$

5.4 KKT Optimality for Discovery

Definition 5.8 (Discovery Problem). *The process discovery problem is:*

find $N^* \in \mathcal{N}$ such that $\text{fit}(N^*, L) \geq \bar{f}$, $\text{prec}(N^*, L) \geq \bar{p}$, and $\text{gen}(N^*, L) + \text{simpl}(N^*)$ is maximised,

where $\mathcal{N}$ is the space of WF-nets (parameterised by arc sets) and $\bar{f}, \bar{p} \in (0, 1)$ are prescribed lower bounds.

Proposition 5.9 (KKT Stationarity Conditions). *At a locally optimal net N^* , the KKT conditions are:*

$$\nabla_N(\text{gen} + \text{simpl})(N^*) = \lambda_f \nabla_N \text{fit}(N^*, L) + \lambda_p \nabla_N \text{prec}(N^*, L),$$

with $\lambda_f, \lambda_p \geq 0$ and complementary slackness: $\lambda_f(\text{fit}(N^*, L) - \bar{f}) = 0$, $\lambda_p(\text{prec}(N^*, L) - \bar{p}) = 0$.

Chapter 6

Evidence Algebra

Algebra is the offer made by the devil to the mathematician: the devil says I will give you this powerful machine, and it will answer any question you like. All you need to do is give me your soul.

Michael Atiyah

6.1 Evidence States as a Poset

Definition 6.1 (Evidence State Poset). The *evidence state poset* $(\mathcal{S}, \leq)$ is the set $\mathcal{S} = \{\perp, \text{Admitted}, \text{Witnessed}, \text{Refused}, \top\}$ with the partial order:

$$\perp \leq \text{Admitted} \leq \text{Witnessed} \leq \top, \quad \perp \leq \text{Refused} \leq \top.$$

The elements Witnessed and Refused are incomparable.

Definition 6.2 (Witnessing Function). A *witnessing function* is a monotone map $w : \mathcal{S} \rightarrow \{0, 1\}$ with $w(\text{Witnessed}) = 1$ and $w(\text{Refused}) = 0$.

6.2 The Evidence Algebra

Definition 6.3 (Evidence Algebra). Let R be a commutative ring (typically $R = \mathbb{Z}$). The *evidence algebra* is the graded R -algebra

$$\mathfrak{E} = \bigoplus_{k=0}^K \mathfrak{E}_k,$$

where:

- $\mathfrak{E}_0 = R$ (base ring, scalars);

- $\mathfrak{E}_k$ (for $k \geq 1$) is the free R -module spanned by artefact evidence tokens $\{e_{k,j}\}$ at lifecycle stage k ;
- Multiplication $\mathfrak{E}_i \times \mathfrak{E}_j \rightarrow \mathfrak{E}_{i+j}$ is defined on generators by the *composition rule*: $e_{i,\alpha} \cdot e_{j,\beta} = e_{i+j,\alpha \otimes \beta}$ (tensor product of causal labels), extended bilinearly.

Theorem 6.4 (Graded Algebra Structure). $(\mathfrak{E}, +, \cdot)$ is a graded R -algebra:

1. $(\mathfrak{E}, +)$ is an abelian group (direct sum of modules).
2. Multiplication is associative and R -bilinear.
3. $\mathfrak{E}_i \cdot \mathfrak{E}_j \subseteq \mathfrak{E}_{i+j}$.
4. The identity element $1 \in \mathfrak{E}_0 = R$ satisfies $1 \cdot e = e$ for all $e \in \mathfrak{E}$.

Proof. (1) Each $\mathfrak{E}_k$ is a free R -module; direct sums of modules are modules.

(2) Associativity: $(e_{i,\alpha} \cdot e_{j,\beta}) \cdot e_{k,\gamma} = e_{i+j,\alpha \otimes \beta} \cdot e_{k,\gamma} = e_{i+j+k,(\alpha \otimes \beta) \otimes \gamma} = e_{i,\alpha} \cdot e_{j+k,\beta \otimes \gamma} = e_{i,\alpha} \cdot (e_{j,\beta} \cdot e_{k,\gamma})$, using associativity of the tensor product on labels. R -bilinearity is immediate from the definition.

(3) The grading condition $\mathfrak{E}_i \cdot \mathfrak{E}_j \subseteq \mathfrak{E}_{i+j}$ follows directly from the multiplication rule on generators.

(4) $1 = e_{0,\emptyset} \in \mathfrak{E}_0 = R$; $e_{0,\emptyset} \cdot e_{k,\alpha} = e_{k,\emptyset \otimes \alpha} = e_{k,\alpha}$ since $\emptyset \otimes \alpha = \alpha$. $\square$

6.3 Filtration

Definition 6.5 (Evidence Filtration). Define $F_k = \bigoplus_{j \geq k} \mathfrak{E}_j$ (the *upper* filtration). Then $\mathfrak{E} = F_0 \supseteq F_1 \supseteq F_2 \supseteq \dots$ and $F_i \cdot F_j \subseteq F_{i+j}$. The *associated graded algebra* $\text{gr}(\mathfrak{E}) \cong \mathfrak{E}$ (trivially, since the filtration is already induced by the grading).

6.4 Lifecycle Morphism Theorem

Definition 6.6 (Lifecycle Morphism). A *lifecycle morphism* is an R -algebra homomorphism $\phi : \mathfrak{E} \rightarrow \mathfrak{E}$ satisfying:

- (M1) $\phi(\mathfrak{E}_k) \subseteq \mathfrak{E}_k$ (grade-preserving);
- (M2) $\phi(e) \geq e$ in the state poset for all generators e (*monotone promotion*);
- (M3) $\phi \circ \phi = \phi$ (*idempotence* — already-witnessed artefacts are not re-witnessed);
- (M4) $\phi(\text{Refused}) = \text{Refused}$ (*refusal absorption* — refused artefacts cannot be promoted);

(M5) For any $e, e' \in \mathfrak{E}_k$: $\phi(e \cdot e') = \phi(e) \cdot \phi(e')$ (*composition compatibility*).

Theorem 6.7 (Lifecycle Morphism Extension). *Any monotone grade-preserving map $\phi_0 : \mathcal{S} \rightarrow \mathcal{S}$ satisfying (M3) and (M4) extends uniquely to a lifecycle morphism $\hat{\phi} : \mathfrak{E} \rightarrow \mathfrak{E}$.*

Proof. Existence. On generators $e_{k,\alpha}$, define $\hat{\phi}(e_{k,\alpha}) = \phi_0(s_\alpha) \cdot e_{k,\alpha}$, where $s_\alpha \in \mathcal{S}$ is the state of the generator. Extend by R -linearity: for $x = \sum_\alpha r_\alpha e_{k,\alpha}$, $\hat{\phi}(x) = \sum_\alpha r_\alpha \hat{\phi}(e_{k,\alpha})$.

To verify (M5): $\hat{\phi}(e_{i,\alpha} \cdot e_{j,\beta}) = \hat{\phi}(e_{i+j,\alpha \otimes \beta}) = \phi_0(s_{\alpha \otimes \beta}) e_{i+j,\alpha \otimes \beta}$. By the tensor product rule $s_{\alpha \otimes \beta} = s_\alpha \otimes s_\beta$ and the monoidal structure of ϕ_0 : $\phi_0(s_\alpha) \cdot \phi_0(s_\beta) = \phi_0(s_\alpha \otimes s_\beta) = \phi_0(s_{\alpha \otimes \beta})$, so $\hat{\phi}(e_{i,\alpha} \cdot e_{j,\beta}) = \hat{\phi}(e_{i,\alpha}) \cdot \hat{\phi}(e_{j,\beta})$.

Idempotence (M3): $\hat{\phi}(\hat{\phi}(x)) = \hat{\phi}(\phi_0(s) \cdot e) = \phi_0(\phi_0(s)) \cdot e = \phi_0(s) \cdot e = \hat{\phi}(x)$ by idempotence of ϕ_0 .

Refusal absorption (M4): $\hat{\phi}(\text{Refused}) = \phi_0(\text{Refused}) = \text{Refused}$.

Uniqueness. Any R -algebra morphism extending ϕ_0 must map generators by $e_{k,\alpha} \mapsto \phi_0(s_\alpha) \cdot e_{k,\alpha}$ (by grade-preservation and R -linearity), which uniquely determines $\hat{\phi}$. $\square$

6.5 Refusal Cokernel

Definition 6.8 (Refusal Cokernel). The *refusal ideal* $I_{\text{ref}} = \{e \in \mathfrak{E} \mid w(e) = 0\}$ is the R -submodule generated by all refused artefact tokens. The *refusal cokernel* is the quotient $\mathfrak{E}/I_{\text{ref}}$.

Proposition 6.9 (Cokernel–Enum Correspondence). *The generators of I_{ref} as an R -module correspond bijectively to the named refusal variants in the `wasm4pm-compact` library:*

$$\begin{aligned} e_{\text{MissingFinalMarking}} &\leftrightarrow \text{PetriRefusal}::\text{MissingFinalMarking}, \\ e_{\text{ObjectTypeNotPreserved}} &\leftrightarrow \text{PetriRefusal}::\text{ObjectTypeNotPreserved}, \\ e_{\text{EmptyLinks}} &\leftrightarrow \text{OcelRefusal}::\text{EmptyEventObjectLinks}, \\ e_{\text{DanglingLink}} &\leftrightarrow \text{OcelRefusal}::\text{DanglingEventObjectLink}. \end{aligned}$$

Proof. Each refusal variant names a specific violated law (Definitions 3.8 and 4.8). The generator e_ρ for refusal reason ρ spans the R -submodule corresponding to all artefacts refused for reason ρ . The bijection maps law labels to module generators. $\square$

6.6 Hopf Algebra Structure

Proposition 6.10 (Bialgebra Extension). $\mathfrak{E}$ admits a coproduct $\Delta : \mathfrak{E} \rightarrow \mathfrak{E} \otimes_R \mathfrak{E}$ defined on generators by

$$\Delta(e_{k,\alpha}) = \sum_{i+j=k} e_{i,\alpha_1} \otimes e_{j,\alpha_2},$$

where $\alpha = \alpha_1 \otimes \alpha_2$ is the canonical causal decomposition. With counit $\varepsilon : \mathfrak{E} \rightarrow R$ given by projection onto $\mathfrak{E}_0$, $(\mathfrak{E}, \mu, \eta, \Delta, \varepsilon)$ is a graded bialgebra.

Proof sketch. Coassociativity $(\Delta \otimes \text{id}) \circ \Delta = (\text{id} \otimes \Delta) \circ \Delta$ follows from associativity of the causal label decomposition. The counit axioms hold by the definition of $\mathfrak{E}_0 = R$. Compatibility of product and coproduct follows from bilinearity. $\square$

Remark 6.11. A full Hopf algebra structure would require an antipode $S : \mathfrak{E} \rightarrow \mathfrak{E}$. The natural candidate is $S(e_{k,\alpha}) = (-1)^k e_{k,\alpha^{\text{op}}}$, where α^{op} reverses the causal order. Verifying that S satisfies the antipode axiom in the presence of refused artefacts (where e_{Refused} must satisfy $S(e_{\text{Refused}}) = e_{\text{Refused}}$) is deferred to Open Problem [12.3](#).

Chapter 7

Soundness–Completeness Duality

Fitness without precision is like a map that shows all roads but includes roads that do not exist.

W. M. P. van der Aalst, *Process Mining* (2016)

7.1 Preliminary Lemmas

Lemma 7.1 (Replay Completeness). *Let N be a WF-net and L an event log. $\text{fit}(N, L) = 1$ if and only if every trace $\sigma \in \text{supp}(\mu_\lambda)$ can be replayed on N without missing or remaining tokens.*

Proof. ($\Rightarrow$) If $\text{fit}(N, L) = 1$, then by the integral formula (5.1),

$$1 = \int \text{fit}_{\text{tr}}(\sigma, N) d\mu_\lambda(\sigma) \leq \int 1 d\mu_\lambda(\sigma) = 1.$$

Since the integrand is ≤ 1 everywhere and the integral equals 1, the integrand must equal 1 μ_λ -almost surely. Hence $\text{fit}_{\text{tr}}(\sigma, N) = 1$ for μ_λ -almost all σ , i.e., for all $\sigma \in \text{supp}(\mu_\lambda)$.

($\Leftarrow$) If $\text{fit}_{\text{tr}}(\sigma, N) = 1$ for all $\sigma \in \text{supp}(\mu_\lambda)$, then the integral equals 1. $\square$

Lemma 7.2 (Enabled-Set Coverage). *If $\text{fit}(N, L) = 1$, then for μ_λ -a.e. σ and every marking M_j reached during replay of σ :*

$$\text{en}(M_j) \supseteq \{\text{activities enabled by the log at position } j\}.$$

Proof. $\text{fit} = 1$ implies zero missing tokens (Lemma 7.1). Zero missing tokens means that whenever the log requires transition t_j , that transition is enabled in M_{j-1} without artificial token injection. Therefore $t_j \in \text{en}(M_{j-1})$, giving the required coverage. $\square$

Lemma 7.3 (Free-Choice Exclusion). *In a free-choice WF-net N , if $M[t\rangle$ and $p \in \bullet t$, then $\text{en}(M) \cap \{t' \mid p \in \bullet t'\} = \{t\}$.*

Proof. By the free-choice property (Definition 3.10), if $p \in \bullet t \cap \bullet t'$ then $\bullet t = \bullet t'$. The enabling condition $M[t]$ requires $M(p) \geq W(p, t)$ for all $p \in \bullet t = \bullet t'$, so $M[t]$ simultaneously. But firing either t or t' consumes the token from p , leaving $M(p) - 1$ (assuming unit weights), which may prevent further enabling. $\square$

7.2 Soundness–Completeness Duality Theorem

Theorem 7.4 (Soundness–Completeness Duality). *Let N be a free-choice WF-net and L an event log with $\text{supp}(\mu_\lambda) \neq \emptyset$. Then:*

$$\text{fit}(N, L) = 1 \iff \text{prec}(N, L) = 1.$$

Proof. ($\Rightarrow$) **Fitness 1 implies Precision 1.**

By Lemma 7.1, $\text{fit}(N, L) = 1$ means every log trace replays without missing or remaining tokens. Fix any $\sigma \in \text{supp}(\mu_\lambda)$ and any position j in the replay. Let M_j be the marking after j steps.

By Lemma 7.2, $\text{en}(M_j)$ contains precisely the transitions needed to replay σ from position j . We claim $\text{en}(M_j) \subseteq \text{Activities}(L)$.

Suppose for contradiction that there exists $t \in \text{en}(M_j)$ with $\text{label}(t) \notin \text{Activities}(L)$. Let $p \in \bullet t$. By the free-choice property and Lemma 7.3, the token at p is exclusively available to transitions sharing $\bullet t$. But t is not in the log, so firing t would consume a token that the log trace requires for a subsequent step, creating a remaining-token deficit — contradicting $\text{fit} = 1$.

Hence $\text{en}(M_j) \subseteq \text{Activities}(L)$ for μ_λ -a.e. σ and all positions j . Therefore

$$\text{prec}(N, L) = \int \frac{|\text{en}(M_\sigma) \cap \text{Activities}(L)|}{|\text{en}(M_\sigma)|} d\mu_\lambda(\sigma) = \int 1 d\mu_\lambda(\sigma) = 1.$$

($\Leftarrow$) **Precision 1 implies Fitness 1.**

$\text{prec}(N, L) = 1$ means that for μ_λ -a.e. σ and at the final replay marking M_σ :

$$|\text{en}(M_\sigma) \cap \text{Activities}(L)| = |\text{en}(M_\sigma)|.$$

This implies every enabled transition at every replay step corresponds to an observed activity. In particular, the replay never visits a state where a required activity is not enabled (since the model enables only activities in the log, and the log requires those activities). Therefore no missing tokens are needed, giving $\text{fit}_{\text{tr}}(\sigma, N) = 1$ for μ_λ -a.e. σ , hence $\text{fit}(N, L) = 1$. $\square$

7.3 Corollaries

Corollary 7.5 (Soundness–Completeness Correspondence). *For a free-choice WF-net N and log L :*

$$\text{fit}(N, L) = 1 \text{ and } N \text{ is sound} \iff N \text{ is the unique minimal sound model for } L.$$

Proof. $(\Rightarrow)$ $\text{fit} = 1$ implies every log trace is in $\mathcal{L}(N)$; soundness ensures N has no dead transitions and terminates properly. Any strictly smaller WF-net would fail to replay some trace, contradicting $\text{fit} = 1$.

$(\Leftarrow)$ The unique minimal sound model for L by definition accepts exactly the traces in L , giving $\text{fit} = \text{prec} = 1$. $\square$

Corollary 7.6 (Non-Free-Choice Failure). *There exists a WF-net N that is not free-choice, an event log L , and values $0 < \alpha < 1$ such that $\text{fit}(N, L) = 1$ but $\text{prec}(N, L) = \alpha < 1$.*

Constructive proof. Let $P = \{i, p_1, p_2, p_3, o\}$, $T = \{t_a, t_b, t_c, t_d\}$, with arcs:

- $i \rightarrow t_a, t_a \rightarrow p_1, t_a \rightarrow p_2$;
- $p_1 \rightarrow t_b, t_b \rightarrow p_3$;
- $p_1 \rightarrow t_c, t_c \rightarrow p_3$;
- $p_2 \rightarrow t_b$; (shared input: $p_2 \in \bullet t_b$)
- $p_3 \rightarrow t_d, t_d \rightarrow o$.

Here t_b has inputs $\{p_1, p_2\}$ and t_c has input $\{p_1\}$ — they share p_1 but $\bullet t_b \neq \bullet t_c$, violating the free-choice property.

Let $L = \{a b d\}^N$ (all traces consist of a , then b , then d). Token replay on N succeeds without missing tokens ($\text{fit} = 1$). However, after firing t_a , both t_b and t_c are enabled at $M = \{p_1^1, p_2^1\}$: $\text{en}(M) = \{t_b, t_c\}$ but only $t_b \in \text{Activities}(L)$. Hence $\text{prec} = 1/2 < 1$. $\square$

Corollary 7.7 (Uniqueness of the Perfect-Quality Point). *For a free-choice WF-net N and log L , the point $(\text{fit}(N, L), \text{prec}(N, L)) = (1, 1)$ is the unique element of $\{1\} \times [0, 1] \cap [0, 1] \times \{1\}$.*

Proof. Immediate from Theorem 7.4: the two sets $\{\text{fit} = 1\}$ and $\{\text{prec} = 1\}$ intersect in the single point $(1, 1)$. $\square$

Chapter 8

Object-Centric Petri Nets

The challenge is to move from a single case perspective to a setting where multiple objects of different types interact in a structured way.

A. Berti and W. M. P. van der Aalst, *OCPN* (2020)

8.1 Definition and Structure

Definition 8.1 (Object-Centric Petri Net). An *Object-Centric Petri Net* (OCPN) is a triple $\mathcal{N} = (N, pt, ot)$ where:

- $N = (P, T, F, W)$ is a Petri net;
- $pt : P \rightarrow \mathcal{Q} \cup \{\tau\}$ is the *place type function* (τ denotes untyped places);
- $ot : F \rightarrow \{\text{regular}, \text{variable}\}$ is the *arc type function*, marking arcs as regular or variable-arc.

A *variable arc* (p, t) or (t, p) carries a *set* (rather than single token) of objects from the place type of p .

Definition 8.2 (Binding). A *binding* for transition $t \in T$ in $\mathcal{N}$ is a function

$$\beta : P_t^\bullet \cup {}^\bullet P_t \rightarrow \bigcup_{q \in \mathcal{Q}} \mathcal{P}_{\text{fin}}(\mathcal{O}_q),$$

where $\mathcal{O}_q = \{o \in \mathcal{O} \mid \text{type}(o) = q\}$, satisfying:

1. *Type consistency*: $\beta(a) \subseteq \mathcal{O}_{pt(p(a))}$ for each arc a incident to t , where $p(a)$ is the place of a .
2. *Regular arc constraint*: if $ot(a) = \text{regular}$, $|\beta(a)| = 1$.
3. *Variable arc constraint*: if $ot(a) = \text{variable}$, $|\beta(a)| \geq 1$.

8.2 Object-Centric Markings and Firing

Definition 8.3 (Object-Centric Marking). An *object-centric marking* of $\mathcal{N}$ is a function $M : P \rightarrow \mathcal{B}(\mathcal{O})$ such that $M(p) \subseteq \mathcal{B}(\mathcal{O}_{pt(p)})$ for all typed places p .

Definition 8.4 (Enabled Binding and Firing). A binding β for t is *enabled* at M if for all input arcs (p, t) :

$$\beta((p, t)) \leq M(p) \quad (\text{as multisets}).$$

The *successor marking* M' after firing t with binding β is:

$$M'(p) = M(p) - \beta((p, t)) + \beta((t, p)) \quad \text{for each } p \in P,$$

where $\beta((p, t)) = \emptyset$ if $(p, t) \notin F$ (and symmetrically).

8.3 Type Preservation

Theorem 8.5 (Type Preservation). Let $\mathcal{N} = (N, pt, ot)$ be an OCPN. For any enabled binding β and successor marking M' produced by firing t with β :

$$\forall p \in P, \forall o \in \lfloor M'(p) \rfloor, \text{type}(o) = pt(p).$$

Proof. By induction on the firing sequence length.

Base case ($k = 0$): The initial marking M_0 is assumed well-typed (part of the model specification).

Inductive step: Assume M is well-typed. After firing t with binding β :

$$M'(p) = M(p) - \beta((p, t)) + \beta((t, p)).$$

For $o \in \lfloor M(p) \rfloor$: $\text{type}(o) = pt(p)$ by inductive hypothesis.

For $o \in \lfloor \beta((t, p)) \rfloor$ (newly added objects): by binding type consistency (Definition 8.2, condition 1), $\beta((t, p)) \subseteq \mathcal{O}_{pt(p)}$, so $\text{type}(o) = pt(p)$.

Removed objects ($o \in \lfloor \beta((p, t)) \rfloor$) are subtracted from $M(p)$; their types remain correct by inductive hypothesis.

Hence $M'(p)$ contains only objects of type $pt(p)$. □

8.4 Binding Existence Theorem

Theorem 8.6 (Binding Existence). Let $\mathcal{N}$ be an OCPN, M an object-centric marking, and $t \in T$ a transition. If for every input place $p \in \bullet t$ there exists at least one object $o \in \lfloor M(p) \rfloor$ of type $pt(p)$, then there exists an enabled binding β for t at M .

Proof. Construct a bipartite graph $G = (A \cup B, E)$ where: $A = \{(p, a) \mid p \in \bullet t, \text{ot}((p, t)) = \text{regular}\}$ (regular input arcs); $B = \cup_{p \in \bullet t} \lfloor M(p) \rfloor$ (available objects); $E = \{((p, a), o) \mid \text{type}(o) = pt(p)\}$.

By hypothesis, every vertex in A has at least one neighbour in B . By König's theorem, the bipartite graph has a matching saturating A . This matching defines $\beta(p, t) = \{o\}$ for each regular arc (p, t) , with o the matched object. For variable arcs, set $\beta(p, t) = \lfloor M(p) \rfloor$. The resulting β satisfies all three conditions of Definition 8.2. $\square$

8.5 OCPN Soundness

Definition 8.7 (OCPN Soundness). An OCPN $\mathcal{N}$ is *sound* if for every object type $q \in \mathcal{Q}$, the type projection N_q (the WF-net obtained by restricting N to places of type q and transitions incident to those places) is a sound WF-net (Definition 3.8).

Theorem 8.8 (OCPN Soundness Reduction). *An OCPN $\mathcal{N}$ is sound (Definition 8.7) if and only if for every $q \in \mathcal{Q}$, the type projection $N_q \in \mathbf{WFNet}$ is a sound WF-net.*

Proof. ($\Rightarrow$) By definition of OCPN soundness.

($\Leftarrow$) Suppose every N_q is sound. Consider a token in $\mathcal{N}$ representing an object o of type q . Its lifecycle in $\mathcal{N}$ projects onto a firing sequence of N_q . Soundness of N_q guarantees this sequence reaches the sink, leaves no superfluous tokens, and uses no dead transitions. Since this holds for all q and all objects of each type, the full OCPN execution is sound. $\square$

8.6 Type Preservation as Computational Refusal Law

Proposition 8.9 (Refusal on Type Violation). *If an arc (p, t) in $\mathcal{N}$ carries an object o with $\text{type}(o) \neq pt(p)$, the OCPN must refuse the transition (return `PetriRefusal::ObjectTypeNotPreserved`) rather than proceed with the type-unsafe binding.*

Proof. Allowing a type-unsafe binding would violate Theorem 8.5: the successor marking would contain an object o of incorrect type in place p . The named refusal `ObjectTypeNotPreserved` is the implementation-level encoding of this violation, as per Proposition 6.9. $\square$

Chapter 9

Type-Theoretic Implementation in Rust

The purpose of types is to prevent the occurrence of execution errors during the running of a program.

Luca Cardelli, *Type Systems* (1997)

9.1 Overview and Design Philosophy

The theoretical apparatus developed in Chapters 2-8 demands an implementation substrate that *encodes invariants as types*, so that violations are detected at compile time rather than at runtime. The present chapter demonstrates how the Rust programming language satisfies this demand through three complementary mechanisms:

1. **Const-generic bounds** (§9.3) – encode rational quality thresholds in the type lattice so that profiles comparing incommensurable bounds are rejected by the type checker.
2. **Typestate patterns** (§9.4) – lift the workflow-net soundness lifecycle of Definition 3.8 into the Rust type system via zero-cost phantom-type markers.
3. **Sealed-capability modules** (§9.5) – guarantee that a `SoundnessWitnessed` value cannot be constructed without passing the validation oracle, mirroring the unforgeable proof-term discipline of constructive type theory.

Together these mechanisms instantiate the Curry-Howard correspondence (§9.2) at the level of process-mining law: a *program that type-checks* is a *proof* that no illegal state transition has occurred.

9.2 Curry-Howard Correspondence

Definition 9.1 (Curry-Howard Isomorphism). Let Π be a simply typed λ -calculus extended with dependent product types and let $\mathcal{P}$ be an intuitionistic propositional logic. The *Curry-Howard isomorphism* is the bijection

$$\text{CH} : \text{Types}(\Pi) \xrightarrow{\sim} \text{Propositions}(\mathcal{P})$$

such that

$$\text{CH}(\sigma \rightarrow \tau) = \text{CH}(\sigma) \Rightarrow \text{CH}(\tau), \quad \text{CH}(\sigma \times \tau) = \text{CH}(\sigma) \wedge \text{CH}(\tau),$$

and a term $t : \sigma$ inhabiting type σ corresponds to a proof p of proposition $\text{CH}(\sigma)$.

In the context of the present work the correspondence is instantiated as follows. Let **Sound** denote the proposition

$$\text{Sound}(N) \equiv N \text{ is a sound WF-net} \quad (\text{Definition 3.8}).$$

The Rust type `WfNet<SoundnessWitnessed>` is the corresponding *proof type*; a value of that type is a *proof term* for **Sound**(N). Constructing such a value requires passing through `WfNet::validate()` – the verification oracle – thereby ensuring

$$\vdash t : \text{WfNet}\langle\text{SoundnessWitnessed}\rangle \implies \text{Sound}(N_t),$$

where N_t is the workflow net embedded in t .

9.3 Const-Generic Quality Bounds

9.3.1 The Nightly Type-Law Machinery

Rust’s const-generic feature (stabilised for basic cases in 1.51, extended via `#![feature(generic_const_exprs)]` on nightly) allows type parameters to range over compile-time integer constants. The library exploits this to encode rational quality thresholds as *types* rather than runtime values.

Definition 9.2 (Rational Quality Bound). A *rational quality bound* is a pair $(N, D) \in \mathbb{N}^2$ with $D > 0$ and $N \leq D$, representing the rational number $N/D \in [0, 1]$.

Proposition 9.3 (Compile-Time Rejection). *For any instantiation `Between01<N,D>` with $N > D$ or $D = 0$, the Rust compiler produces a type error at the call site. No runtime check is required.*

Listing 1 Compile-time interval enforcement via const predicates.

```
1  #![feature(generic_const_exprs)]
2
3  /// Marker type whose existence encodes the proposition P =
4      true.
5
6  pub struct Require<const P: bool>;
7
8  /// Sealed trait inhabited only when P = true.
9  pub trait IsTrue {}
10 impl IsTrue for Require<true> {}
11
12 /// A rational in [0,1] encoded at the type level.
13 ///
14 /// Invariants (enforced at compile time):
15 ///     - DEN > 0
16 ///     - NUM <= DEN
17 pub struct Between01<const NUM: u64, const DEN: u64>
18 where
19     Require<{ DEN > 0 }>: IsTrue,
20     Require<{ NUM <= DEN }>: IsTrue,
21 {
22     _private: (),
23 }
24
25 impl<const NUM: u64, const DEN: u64> Between01<NUM, DEN>
26 where
27     Require<{ DEN > 0 }>: IsTrue,
28     Require<{ NUM <= DEN }>: IsTrue,
29 {
30     pub const fn new() -> Self {
31         Self { _private: () }
32     }
33
34     /// Returns the bound as a 64-bit float, computed at
35         compile time.
36     pub const fn value() -> f64 {
37         NUM as f64 / DEN as f64
38     }
39 }
```

Proof. The where-clause `Require<{NUM <= DEN}>: IsTrue` resolves to `Require<false>: IsTrue`, for which no `impl` block exists. Rust’s type-class resolution therefore fails with an unsatisfied trait bound, halting compilation. The argument for $D = 0$ is symmetric. $\square$

9.3.2 Dimension-Specific Newtypes

The four Van der Aalst quality dimensions (Definition 5.6) are distinct concepts and must not be silently interchanged. The implementation defines five distinct struct types – `FitnessConst`, `PrecisionConst`, `F1Const`, `GeneralizationConst`, `SimplicityConst` – each parameterised by the same (N, D) pattern but carrying distinct identities in the type system.

Listing 2 Dimension-specific const-generic quality types.

```

1 macro_rules! quality_const_type {
2     ($Name:ident) => {
3         pub struct $Name<const NUM: u64, const DEN: u64>
4         where
5             Require<{ DEN > 0 }>: IsTrue,
6             Require<{ NUM <= DEN }>: IsTrue,
7         {
8             _private: (),
9         }
10
11         impl<const NUM: u64, const DEN: u64> $Name<NUM, DEN>
12         where
13             Require<{ DEN > 0 }>: IsTrue,
14             Require<{ NUM <= DEN }>: IsTrue,
15         {
16             pub const fn new() -> Self { Self { _private: () } }
17             pub const fn num(&self) -> u64 { NUM }
18             pub const fn den(&self) -> u64 { DEN }
19         }
20     };
21 }
22
23 quality_const_type!(FitnessConst);
24 quality_const_type!(PrecisionConst);
25 quality_const_type!(F1Const);
26 quality_const_type!(GeneralizationConst);
27 quality_const_type!(SimplicityConst);

```

A *quality profile* aggregates all five dimensions:

Remark 9.4. A profile demanding fitness ≥ 1.01 or precision > 1 cannot be *named* in a Rust source file; the type simply does not exist. This is the compile-time analogue of

Listing 3 QualityProfile: a single type encoding five simultaneous lower bounds.

```
1 pub struct QualityProfile<
2     const FN: u64, const FD: u64,
3     const PN: u64, const PD: u64,
4     const F1N: u64, const F1D: u64,
5     const GN: u64, const GD: u64,
6     const SN: u64, const SD: u64,
7 >
8 where
9     Require<{ FD > 0 }>: IsTrue, Require<{ FN <= FD }>: IsTrue,
10    Require<{ PD > 0 }>: IsTrue, Require<{ PN <= PD }>: IsTrue,
11    Require<{ F1D > 0 }>: IsTrue, Require<{ F1N <= F1D }>:
12        IsTrue,
13    Require<{ GD > 0 }>: IsTrue, Require<{ GN <= GD }>: IsTrue,
14    Require<{ SD > 0 }>: IsTrue, Require<{ SN <= SD }>: IsTrue,
15 {
16     pub fitness: FitnessConst<FN, FD>,
17     pub precision: PrecisionConst<PN, PD>,
18     pub f1: F1Const<F1N, F1D>,
19     pub generalization: GeneralizationConst<GN, GD>,
20     pub simplicity: SimplicityConst<SN, SD>,
21 }
```

the algebraic constraint $\text{fit}, \text{prec} \in [0, 1]$ imposed in Definition 5.6.

9.4 Typestate and Soundness Lifecycle

9.4.1 Motivation

Consider the soundness lifecycle of a workflow net (Chapter 3):

$$\text{Unchecked} \xrightarrow{\text{claim_sound}} \text{SoundnessClaimed} \xrightarrow{\text{witness_soundness}} \text{SoundnessWitnessed}.$$

An implementation that stores soundness state in a runtime boolean field permits the following illegal program:

```
1 let mut net = WfNet::new(...);
2 net.soundness = true;           // forgery -- no validation
   occurred
```

The typestate pattern closes this attack surface by making each lifecycle stage a distinct *type*.

9.4.2 Marker Types and PhantomData

Definition 9.5 (Zero-Size Type Marker). A *zero-size type* (ZST) in Rust is a type with no data fields, occupying zero bytes at runtime. A ZST marker S used as a phantom type parameter encodes a predicate P_S about the enclosing struct without affecting its memory layout.

Theorem 9.6 (Typestate Soundness). *Under the above encoding, every value of type `WfNet<SoundnessWitnessed>` was produced by a call to `witness_soundness` on a `WfNet<SoundnessClaimed>`, which in turn was produced by `claim_sound` from a `WfNet<Unchecked>`.*

Proof. The type `WfNet<SoundnessWitnessed>` has no `pub fn new` or `Default` impl. The only constructor that produces it is `witness_soundness`, which takes ownership of a `WfNet<SoundnessClaimed>`. By induction on the constructor chain, no value of type `WfNet<SoundnessWitnessed>` can be produced without traversing the full lifecycle. The Rust ownership model guarantees no aliasing; `PhantomData` ensures the marker is not erased. $\square$

9.5 Sealed-Capability Pattern

9.5.1 Definition

Definition 9.7 (Sealed Capability). A *sealed capability* is a value type T whose sole constructor is `pub(crate)` (or more restrictive), such that:

1. External crates cannot construct T directly.
2. Internal code must invoke a designated *minting function* `mint : Proof  $\rightarrow$  T`, which may perform arbitrary validation before producing the capability.

Proposition 9.8 (Unforgeability). *No external crate can construct a value of type `SoundnessProof` without calling `SoundnessProof::new()`.*

Proof. The struct field `SoundnessProof(wfnet_seal::WfNetSeal)` is a newtype over a module-private type. Rust’s privacy rules prevent external crates from naming `wfnet_seal::WfNetSeal`; therefore no external code can write a struct literal for `SoundnessProof(...)`. Since there is no `Default`, `Clone`, or `Copy` impl, and no other constructor is `pub`, the only valid way to obtain the value is through `SoundnessProof::new()`. $\square$

Listing 4 Typestate markers for WF-net soundness.

```
1 use std::marker::PhantomData;
2
3 /// Private; cannot be named outside this module.
4 struct Unchecked;
5
6 /// Public; can be named as a bound but not constructed
externally.
7 pub struct SoundnessClaimed;
8
9 /// Public; proves that witness_soundness() was called with a
valid proof.
10 pub struct SoundnessWitnessed;
11
12 pub struct WfNet<S = Unchecked> {
13     net: PetriNet,
14     final_marking: Marking,
15     _s: PhantomData<S>,
16 }
17
18 impl WfNet<Unchecked> {
19     pub fn new(net: PetriNet, final_marking: Marking) -> Self {
20         Self { net, final_marking, _s: PhantomData }
21     }
22
23     #[must_use]
24     pub fn validate(&self) -> Result<(), PetriRefusal> {
25         if self.final_marking.is_empty() {
26             return Err(PetriRefusal::MissingFinalMarking);
27         }
28         Ok(())
29     }
30
31     #[must_use]
32     pub fn claim_sound(self) -> WfNet<SoundnessClaimed> {
33         WfNet { net: self.net, final_marking: self.
34             final_marking,
35             _s: PhantomData }
36     }
37 }
38
39 impl WfNet<SoundnessClaimed> {
40     /// Consumes the claimed net and a proof token to produce a
witnessed net.
41     #[must_use]
42     pub fn witness_soundness(self, _proof: SoundnessProof)
43         -> WfNet<SoundnessWitnessed>
44     {
45         WfNet { net: self.net, final_marking: self.
46             final_marking,
47             _s: PhantomData }
48     }
49 }
50
51 impl WfNet<SoundnessWitnessed> {
52     pub fn soundness_state(&self) -> SoundnessState {
53         SoundnessState::Witnessed
54     }
55 }
```

Listing 5 Sealed SoundnessProof with private constructor.

```
1 mod wfnet_seal {
2     /// The seal type is never exported; it is the unforgeable
        token.
3     pub(super) struct WfNetSeal;
4 }
5
6 /// A proof token for WF-net soundness. Cannot be forged
    outside this module.
7 pub struct SoundnessProof(wfnet_seal::WfNetSeal);
8
9 impl SoundnessProof {
10     /// The only minting function. In production this would
        invoke
11     /// the full structural soundness verification algorithm
12     /// (cf. Theorem~\ref{thm:soundness-shortcircuit}).
13     pub(crate) fn new() -> Self {
14         Self(wfnet_seal::WfNetSeal)
15     }
16 }
```

9.5.2 Comparison to Dependent Types

In a dependently typed language such as Coq or Agda, the analogous construct would be an *existential type*:

$$\exists N : \text{WFNet}, \text{Sound}(N).$$

The sealed-capability pattern in Rust approximates this by replacing the constructive proof $\text{Sound}(N)$ with an opaque token `SoundnessProof` whose existence is contingent on having run the verifier. The approximation is sound but not complete: a buggy verifier could mint a `SoundnessProof` for an unsound net. Theorem-proving assistants would detect this at proof-check time; Rust trusts the implementation of `SoundnessProof::new()`.

9.6 Named Refusal Enums

Every validation function in the library returns a named *refusal* rather than a generic error string. This design is motivated by Definition 4.8 (OCEL validation laws) and Lemma 7.1 (replay refusal conditions): a validator that refuses without a name forces the caller to perform string matching, which is fragile and non-composable.

Remark 9.9. The `Display` implementation emits the variant name verbatim. This allows downstream code to pattern-match on `err.to_string()` in log-driven conformance checks without coupling to the enum layout. The convention mirrors the van der Aalst refusal taxonomy: each refusal name is a *law label* (e.g., L_1 - L_4 in Definition 4.8), making

Listing 6 Named refusal enums for Petri-net and OCEL validation.

```
1  /// Reasons a WF-net may be structurally invalid.
2  #[derive(Debug, Clone, PartialEq, Eq)]
3  pub enum PetriRefusal {
4      /// The final marking is empty; soundness condition (S3)
4          fails.
5      MissingFinalMarking,
6      /// An arc references an object type not declared in the
6          OCPN.
7      ObjectTypeNotPreserved,
8  }
9
10 impl fmt::Display for PetriRefusal {
11     fn fmt(&self, f: &mut fmt::Formatter<'_>) -> fmt::Result {
12         match self {
13             Self::MissingFinalMarking => write!(f, "
14                 MissingFinalMarking"),
15             Self::ObjectTypeNotPreserved => write!(f, "
16                 ObjectTypeNotPreserved"),
17         }
18     }
19 }
20
21 /// Reasons an OCEL 2.0 log may violate the well-formedness
21     axioms.
22 #[derive(Debug, Clone, PartialEq, Eq)]
23 pub enum OcelRefusal {
24     /// No event-object links exist; log is uninterpretable.
24     EmptyEventObjectLinks,
25     /// An event-object link references an object id not in the
25     log.
26     DanglingEventObjectLink,
27 }
```

the correspondence between runtime behaviour and formal specification explicit.

9.7 Runtime Fitness Newtype

While const-generic bounds (§9.3) are appropriate for statically known thresholds, many conformance computations produce quality values at runtime. The library provides runtime newtypes:

Listing 7 Runtime Fitness newtype with Option-returning constructor.

```
1 #[derive(Debug, Clone, Copy, PartialEq)]
2 #[repr(transparent)]
3 pub struct Fitness(f64);
4
5 impl Fitness {
6     /// Returns None for NaN, +/-inf, or values outside [0,1].
7     #[must_use]
8     pub fn new(v: f64) -> Option<Self> {
9         if v.is_finite() && v >= 0.0 && v <= 1.0 {
10             Some(Self(v))
11         } else {
12             None
13         }
14     }
15
16     pub fn get(self) -> f64 { self.0 }
17 }
```

The `#[repr(transparent)]` attribute guarantees that `Fitness` has the same memory layout as `f64`, making it a zero-cost abstraction in the sense of Definition 9.5: the newtype wrapper adds no runtime overhead while enforcing the invariant $\text{fit} \in [0, 1]$ at every construction site.

9.8 Tests as Existence Proofs

Under the Curry-Howard correspondence (Definition 9.1), a Rust unit test that compiles and passes is a *constructive proof* that the encoded property holds for the exhibited witness. The following test proves the full tpestate lifecycle (Theorem 9.6):

These tests satisfy the Van der Aalst *evidence-first* principle: they do not assert on internal state via mocking frameworks; they exercise the public API boundary and observe the resulting tpestate transition. In the terminology of Chapter 5, the test trace is the event log and the tpestate lifecycle diagram is the reference model; the test constitutes a conformance replay with $\text{fitness} = 1$.

Listing 8 Typestate lifecycle test as a constructive existence proof.

```
1 #[cfg(test)]
2 mod tests {
3     use super::*;
4
5     /// Proves: there exists a WfNet that can traverse
6     Unchecked ->
7     /// SoundnessClaimed -> SoundnessWitnessed without
8     panicking.
9     #[test]
10    fn soundness_proof_mints_and_advances_to_witnessed() {
11        let proof = SoundnessProof::new();
12        let witnessed = WfNet::new(/* ... */)
13            .claim_sound()
14            .witness_soundness(proof);
15        assert_eq!(witnessed.soundness_state(), SoundnessState
16            ::Witnessed);
17    }
18
19    /// Proves: the refused constructor carries its diagnostic
20    value.
21    #[test]
22    fn refused_constructor_carries_value_for_diagnostics() {
23        let refused = Evidence::<_, Refused, Ocel20>::refused("
24            boundary-reject");
25        assert_eq!(*refused.as_refused_value(), "boundary-
26            reject");
27    }
28 }
```

9.9 Chapter Summary

This chapter has demonstrated that the formal theory of Chapters 2-8 maps faithfully to a type-safe Rust implementation via three interlocking patterns: const-generic bounds encode algebraic range constraints as types; typestate patterns lift soundness lifecycles into the type lattice; and sealed capabilities provide unforgeable proof tokens without a dependent type system. The resulting library provides strong correctness guarantees while remaining a zero-cost abstraction – all extra type structure is erased at compile time with no runtime penalty.

The next chapter (10) shows how Declare’s temporal logic constraints integrate into this type-theoretic framework.

Chapter 10

Declare: Constraint-Based Process Specification

The notion of flexibility in process management has long been advocated, but formalising it rigorously in terms of temporal logic remains a challenge.

W. M. P. van der Aalst, *Process Mining* (2016)

10.1 Motivation and Historical Context

The imperative process models of Chapter 3 specify *what must happen*: every run of the net is a sequence of firings consistent with the flow relation. Practical processes, however, often admit high degrees of *flexibility*: workers choose the order of activities subject only to a set of declarative rules.

Pesic and van der Aalst introduced the *Declare* language [4] to formalise this intuition. Declare specifies processes via a finite set of *constraints*, each expressed as a Linear Temporal Logic (LTL) formula over the activity alphabet. A trace is *conforming* if and only if it satisfies every constraint.

Definition 10.1 (Trace Semantics). Let Σ be an activity alphabet and $\sigma = a_1a_2 \cdots a_n \in \Sigma^*$ a finite trace. For $i \in \{1, \dots, n\}$, write $\sigma, i \models \phi$ for the standard LTL satisfaction

relation over finite words, with the *finite-trace* semantics:

$$\begin{array}{ll}
\sigma, i \models a & \text{iff } a_i = a, \\
\sigma, i \models \neg\phi & \text{iff } \sigma, i \not\models \phi, \\
\sigma, i \models \phi \wedge \psi & \text{iff } \sigma, i \models \phi \text{ and } \sigma, i \models \psi, \\
\sigma, i \models \mathbf{X}\phi & \text{iff } i < n \text{ and } \sigma, i + 1 \models \phi, \\
\sigma, i \models \phi \mathbf{U} \psi & \text{iff } \exists j \geq i, \sigma, j \models \psi \text{ and } \forall k \in [i, j), \sigma, k \models \phi.
\end{array}$$

We write $\sigma \models \phi$ for $\sigma, 1 \models \phi$.

10.2 LTL Foundations

Definition 10.2 (LTL Syntax). The set $\text{LTL}(\Sigma)$ of *LTL formulas over* Σ is the smallest set satisfying:

1. $\mathbf{true} \in \text{LTL}(\Sigma)$.
2. $a \in \Sigma \Rightarrow a \in \text{LTL}(\Sigma)$.
3. $\phi \in \text{LTL}(\Sigma) \Rightarrow \neg\phi \in \text{LTL}(\Sigma)$.
4. $\phi, \psi \in \text{LTL}(\Sigma) \Rightarrow (\phi \wedge \psi), (\phi \vee \psi), (\phi \Rightarrow \psi) \in \text{LTL}(\Sigma)$.
5. $\phi, \psi \in \text{LTL}(\Sigma) \Rightarrow \mathbf{X}\phi, \mathbf{G}\phi, \mathbf{F}\phi, \phi \mathbf{U} \psi \in \text{LTL}(\Sigma)$.

Derived operators: $\mathbf{G}\phi \equiv \neg(\mathbf{true} \mathbf{U} \neg\phi)$; $\mathbf{F}\phi \equiv \mathbf{true} \mathbf{U} \phi$.

Definition 10.3 (Declare Constraint). A *Declare constraint* over alphabet Σ is a pair $(T, \vec{a})$ where T is a *template* (Definition 10.5) and $\vec{a} \in \Sigma^{\text{ar}(T)}$ is a tuple of activities matching the template's arity.

Definition 10.4 (Declare Model). A *Declare model* is a pair $(\Sigma, \mathcal{C})$ where $\mathcal{C}$ is a finite set of Declare constraints over Σ . The *language* of the model is

$$\mathcal{L}(\Sigma, \mathcal{C}) = \left\{ \sigma \in \Sigma^+ \mid \forall (T, \vec{a}) \in \mathcal{C}, \sigma \models \phi_T(\vec{a}) \right\}.$$

10.3 Declare Template Catalogue

Definition 10.5 (Declare Template). A *Declare template* T is a function schema $T : \Sigma^{\text{ar}(T)} \rightarrow \text{LTL}(\Sigma)$ mapping activity parameters to a concrete LTL formula. We classify templates by arity and semantic polarity.

Table 10.1: Declare template catalogue. $a, b \in \Sigma$; $\mathbf{G}, \mathbf{F}, \mathbf{X}, \mathbf{U}$ are standard LTL operators.

Template	LTL formula	Polarity	Arity
<i>Unary existence templates</i>			
EXISTENCE(a)	$\mathbf{F} a$	pos	1
ABSENCE(a)	$\mathbf{G} \neg a$	neg	1
INIT(a)	a	pos	1
EXISTENCE2(a)	$\mathbf{F}(a \wedge \mathbf{X} \mathbf{F} a)$	pos	1
EXISTENCE3(a)	$\mathbf{F}(a \wedge \mathbf{X}(\mathbf{F}(a \wedge \mathbf{X} \mathbf{F} a)))$	pos	1
ABSENCE2(a)	$\neg \mathbf{F}(a \wedge \mathbf{X} \mathbf{F} a)$	neg	1
ABSENCE3(a)	$\neg \mathbf{F}(a \wedge \mathbf{X}(\mathbf{F}(a \wedge \mathbf{X} \mathbf{F} a)))$	neg	1
<i>Binary positive relation templates</i>			
RESPONDEDEXISTENCE(a, b)	$\mathbf{F} a \Rightarrow \mathbf{F} b$	pos	2
COEXISTENCE(a, b)	$\mathbf{F} a \Leftrightarrow \mathbf{F} b$	pos	2
RESPONSE(a, b)	$\mathbf{G}(a \Rightarrow \mathbf{F} b)$	pos	2
PRECEDENCE(a, b)	$\mathbf{F} b \Rightarrow a \mathbf{U} b$	pos	2
SUCCESSION(a, b)	$\mathbf{G}(a \Rightarrow \mathbf{F} b) \wedge (\mathbf{F} b \Rightarrow a \mathbf{U} b)$	pos	2
ALTERNATERESPONSE(a, b)	$\mathbf{G}(a \Rightarrow \mathbf{X}(\neg a \mathbf{U} b))$	pos	2
ALTERNATEPRECEDENCE(a, b)	$\mathbf{G}(b \Rightarrow \mathbf{X}^{-1}(\neg b \mathbf{S} a))$	pos	2
ALTERNATESUCCESSION(a, b)	$\text{ALTRESP}(a, b) \wedge \text{ALTPREC}(a, b)$	pos	2
CHAINRESPONSE(a, b)	$\mathbf{G}(a \Rightarrow \mathbf{X} b)$	pos	2
CHAINPRECEDENCE(a, b)	$\mathbf{G}(b \Rightarrow \mathbf{X}^{-1} a)$	pos	2
CHAINSUCCESSION(a, b)	$\mathbf{G}(a \Leftrightarrow \mathbf{X} b)$	pos	2
<i>Binary negative relation templates</i>			
NOTSUCCESSION(a, b)	$\mathbf{G}(a \Rightarrow \mathbf{G} \neg b)$	neg	2
NOTCHAINSUCCESSION(a, b)	$\mathbf{G}(a \Rightarrow \neg \mathbf{X} b)$	neg	2
NOTCOEXISTENCE(a, b)	$\neg(\mathbf{F} a \wedge \mathbf{F} b)$	neg	2
EXCLUSIVECHOICE(a, b)	$(\mathbf{F} a \vee \mathbf{F} b) \wedge \neg(\mathbf{F} a \wedge \mathbf{F} b)$	mixed	2

Table 10.1 gives the complete catalogue of 22 templates used in the present implementation, together with their LTL translations and semantic roles.

Remark 10.6. EXCLUSIVECHOICE is classified as “mixed” because it contains both a positive component ($\mathbf{F} a \vee \mathbf{F} b$) and a negative one ($\neg(\mathbf{F} a \wedge \mathbf{F} b)$). In the Rust implementation, `DeclareTemplate::is_negative()` returns `false` for EXCLUSIVECHOICE, reflecting that the template overall enforces a selection rather than a prohibition.

10.4 Arity and Polarity Functions

Definition 10.7 (Template Arity). The *arity function* $\text{ar} : \mathcal{T} \rightarrow \{1, 2\}$ assigns to each template T the number of activity parameters it requires:

$$\text{ar}(T) = 1 \text{ for the 7 unary templates,} \quad \text{ar}(T) = 2 \text{ for the 15 binary templates.}$$

Definition 10.8 (Template Polarity). The *polarity predicate* $\text{neg} : \mathcal{T} \rightarrow \{\top, \perp\}$ classifies templates that are primarily prohibitive:

$$\text{neg}(T) = \top \Leftrightarrow T \in \{\text{ABSENCE}, \text{ABSENCE2}, \text{ABSENCE3}, \text{NOTCOEXISTENCE}, \text{NOTSUCCESSION}, \text{NOTCHAINRESPONSE}\}$$

The *chain predicate* $\text{chain} : \mathcal{T} \rightarrow \{\top, \perp\}$ is:

$$\text{chain}(T) = \top \Leftrightarrow T \in \{\text{CHAINRESPONSE}, \text{CHAINPRECEDENCE}, \text{CHAINSUCCESSION}, \text{NOTCHAINSUCCESSION}\}$$

10.5 Translation to Workflow Nets

A fundamental result connecting Declare to Chapter 3 is the compilation of each template into a WF-net fragment.

Theorem 10.9 (Declare-to-WF-Net Translation). *For each Declare template T of arity k and any $\vec{a} \in \Sigma^k$, there exists a WF-net $N_{T,\vec{a}}$ such that for every trace σ :*

$$\sigma \models \phi_T(\vec{a}) \iff \sigma \text{ is a firing sequence of } N_{T,\vec{a}}.$$

Proof sketch. The construction proceeds by structural induction on T .

Base cases (unary):

- **EXISTENCE(a):** Construct N with source place p_s , sink place p_e , and a single transition t_a (labelled a) between them. The unique firing sequence t_a witnesses $\mathbf{F} a$.
- **ABSENCE(a):** Add a labelled *silent* place as a “skip” arc bypassing t_a ; block firing of t_a once. Any trace that fires t_a reaches a deadlock before the final marking.

Inductive cases (binary):

- **RESPONSE**(a, b): Introduce a *pending* counter place p_{pend} . Firing t_a deposits a token in p_{pend} ; only t_b can consume it. The final marking requires p_{pend} to be empty; hence every trace ending in the final marking has fired t_b after every t_a .
- **CHAINRESPONSE**(a, b): Add an obligation place p_{obl} that must be emptied by the immediately next transition. Any transition other than t_b in state $p_{\text{obl}} = 1$ leads to deadlock.

The remaining templates are handled by combining these gadgets. A formal proof via Petri-net bisimulation is given by Pesic [4], Theorem 4.1. $\square$

Corollary 10.10 (Decidability of Declare Conformance). *Given a Declare model $(\Sigma, \mathcal{C})$ and a trace $\sigma \in \Sigma^*$, the conformance problem $\sigma \in \mathcal{L}(\Sigma, \mathcal{C})$ is decidable in time $O(|\sigma| \cdot |\mathcal{C}|)$.*

Proof. By Theorem 10.9, each constraint translates to a WF-net of constant size (bounded by the template). Token replay (Algorithm ??) over a WF-net of size k takes $O(|\sigma| \cdot k)$ time. Summing over $|\mathcal{C}|$ constraints gives the claimed bound. $\square$

10.6 Object-Centric Declare

Classical Declare operates over a single case perspective; OCEL 2.0 demands extension to the multi-object setting.

Definition 10.11 (Object-Centric Declare Constraint). Let $\mathcal{Q}$ be a set of object types. An *object-centric Declare constraint* is a triple (C, Λ, s) where:

- $C \in \mathcal{C}$ is a classical Declare constraint.
- $\Lambda \subseteq \mathcal{Q}$ is a non-empty set of object types over which the constraint ranges.
- $s \in \{\text{Multi}, \text{Sync}\}$ is the *scope selector*: **Multi** applies C independently to each object of types in Λ ; **Sync** requires C to hold on *synchronized* multi-type traces.

Definition 10.12 (Synchronized Trace). Let $\Lambda = \{q_1, \dots, q_m\}$ with $m \geq 2$. A *synchronized trace* for Λ is a trace σ over $\Sigma \times \mathcal{Q}$ such that for every pair of consecutive events (a, q_i) and (b, q_j) in σ involving distinct object types $q_i \neq q_j$, there exists a synchronisation event $(c, q_i, q_j) \in \sigma$ establishing a causal link between the two.

Proposition 10.13 (Scope Mismatch Refusal). *An **OcDeclareConstraint** with scope selector **Sync** but fewer than two object types is ill-formed. The **validate()** function returns **OcDeclareRefusal::SynchronizationRequiresMultipleTypes**.*

Proof. A synchronisation event requires at least two object types between which to establish the causal link (Definition 10.12). With $|\Lambda| = 1$, no such pair exists, making the scope semantically vacuous. The refusal is therefore a *necessary condition* check, not merely a defensive guard. $\square$

10.7 Implementation in wasm4pm-compat

The `src/declare.rs` module instantiates the above theory.

Remark 10.14. The `Copy` bound on `DeclareTemplate` is required because constraint structs store the template as a direct field, and many conformance-checking loops iterate over constraints without needing ownership transfer. Since all variants are ZSTs (no heap allocation), `Copy` is both semantically correct and free.

10.8 Complexity Analysis

Theorem 10.15 (Declare Conformance Complexity). *For a Declare model with $|\mathcal{C}|$ constraints over alphabet Σ and a log L with N traces of average length $\bar{\ell}$:*

1. Single-trace conformance: $O(\bar{\ell} \cdot |\mathcal{C}|)$.
2. Log-level conformance: $O(N \cdot \bar{\ell} \cdot |\mathcal{C}|)$.
3. Object-centric (MultiObjectScope): *multiply by $|\Lambda|$.*
4. Object-centric (SynchronizedObjectScope): *multiply by $|\Lambda|!$ in the worst case (synchronisation pairing).*

Proof. Claims (1) and (2) follow from Corollary 10.10 by linearity. Claim (3) follows because each independent object-type projection runs single-trace conformance independently. Claim (4) holds because synchronisation requires finding a consistent inter-type event pairing, which is equivalent to solving a matching problem over $|\Lambda|$ types; the worst-case is the permanent of an $|\Lambda| \times |\Lambda|$ matrix, computed in $O(|\Lambda|!)$ by inclusion-exclusion without further approximation. $\square$

10.9 Chapter Summary

This chapter has established the formal semantics of the Declare constraint-based process language in terms of LTL over finite traces, catalogued all 22 standard templates (Table 10.1), proved their translation to WF-nets (Theorem 10.9), and extended the formalism to the object-centric setting (Definition 10.11). The implementation in `wasm4pm-compat` reflects these definitions directly: the arity and polarity methods

Listing 9 Declare template as a Copy enum with arity and polarity methods.

```
1 #[derive(Debug, Clone, Copy, PartialEq, Eq, Hash)]
2 pub enum DeclareTemplate {
3     // Unary
4     Existence, Absence, Init, Existence2, Existence3, Absence2,
5         Absence3,
6     // Binary positive
7     RespondedExistence, CoExistence, Response, Precedence,
8         Succession,
9     AlternateResponse, AlternatePrecedence, AlternateSuccession
10    ,
11    ChainResponse, ChainPrecedence, ChainSuccession,
12    // Binary negative
13    NotSuccession, NotChainSuccession, NotCoExistence,
14        ExclusiveChoice,
15 }
16
17 impl DeclareTemplate {
18     /// Returns 1 for unary templates, 2 for binary templates.
19     pub fn arity(&self) -> usize {
20         match self {
21             Self::Existence | Self::Absence | Self::Init
22             | Self::Existence2 | Self::Existence3
23             | Self::Absence2 | Self::Absence3 => 1,
24             _ => 2,
25         }
26     }
27
28     /// True for templates whose semantics are primarily
29     prohibitive.
30     pub fn is_negative(&self) -> bool {
31         matches!(self,
32             Self::Absence | Self::Absence2 | Self::Absence3
33             | Self::NotCoExistence | Self::NotSuccession
34             | Self::NotChainSuccession
35         )
36     }
37
38     /// True for templates requiring immediate-next-step
39     enforcement.
40     pub fn is_chain(&self) -> bool {
41         matches!(self,
42             Self::ChainResponse | Self::ChainPrecedence
43             | Self::ChainSuccession | Self::NotChainSuccession
44         )
45     }
46 }
```

of `DeclareTemplate` encode Definitions 10.7 and 10.8, and the refusal variants of `OccDeclareRefusal` encode Proposition 10.13.

Chapter 11 will lift the discrete quality metrics of conformance checking into a continuous differential-geometric framework, enabling gradient-based discovery of optimal process models.

Chapter 11

Differential Geometry of Quality Metrics

The arc length of the shortest path between two probability distributions measures how distinguishable they are — and, by extension, how much information separates them.

Shun-ichi Amari, *Information Geometry and Its Applications*
(2016)

11.1 Motivation

The four quality dimensions (fit, prec, gen, simpl) of Chapter 5 take values in the unit cube $[0, 1]^4$. Process discovery can be framed as an *optimisation problem* over this space: find a process model N whose quality vector $q(N)$ satisfies prescribed minima while maximising some aggregate objective.

This framing is standard in the literature [1, 5], but the geometry of $[0, 1]^4$ as a Riemannian manifold has not been fully exploited. The present chapter develops the differential-geometric structure of the quality space, derives geodesic equations connecting any two quality vectors, and shows that gradient descent on the log-likelihood of an event log under a stochastic net model converges to a local optimum of the Lagrangian discovery problem.

11.2 The Quality Manifold

Definition 11.1 (Quality Manifold). The *quality manifold* $\mathcal{M}$ is the open unit hypercube

$$\mathcal{M} = (0, 1)^4$$

equipped with coordinates (fit, prec, gen, simpl) and the smooth structure inherited from $\mathbb{R}^4$.

Remark 11.2. We work on the open interior $(0, 1)^4$ rather than the closed cube $[0, 1]^4$ to avoid boundary singularities in the Riemannian metric defined below. In practice, a model achieving perfect fitness or precision lies at the boundary; such models are limiting cases and can be treated by continuity.

Definition 11.3 (Fisher–Rao Metric on $\mathcal{M}$). Identify each quality vector $q = (\text{fit}, \text{prec}, \text{gen}, \text{simpl}) \in \mathcal{M}$ with a product of four independent Bernoulli distributions $\text{Ber}(\text{fit}) \otimes \text{Ber}(\text{prec}) \otimes \text{Ber}(\text{gen}) \otimes \text{Ber}(\text{simpl})$. The *Fisher–Rao metric* on $\mathcal{M}$ is the Fisher information metric of this statistical model:

$$g_{ij}(q) = \mathbb{E}_q \left[\frac{\partial \log p(x; q)}{\partial q^i} \frac{\partial \log p(x; q)}{\partial q^j} \right],$$

which, for the product-Bernoulli model, yields a diagonal metric tensor:

$$g = \text{diag} \left(\frac{1}{\text{fit}(1 - \text{fit})}, \frac{1}{\text{prec}(1 - \text{prec})}, \frac{1}{\text{gen}(1 - \text{gen})}, \frac{1}{\text{simpl}(1 - \text{simpl})} \right).$$

Remark 11.4. The Fisher–Rao metric is the unique (up to scaling) Riemannian metric on the statistical manifold that is invariant under sufficient statistics and information-preserving transformations [9]. Its use here is motivated by the interpretation of each quality dimension as the success probability of a Bernoulli trial: fitness is the probability that a random trace is replayed without missing tokens; precision is the probability that a randomly chosen enabled transition corresponds to an observed activity.

11.3 Geodesic Equations

Proposition 11.5 (Christoffel Symbols). *For the diagonal metric $g_{ij} = h_i(q^i) \delta_{ij}$ with $h_i(x) = [x(1 - x)]^{-1}$, the only non-zero Christoffel symbols are*

$$\Gamma_{ii}^i = \frac{1}{2} \frac{h'_i(q^i)}{h_i(q^i)} = \frac{2q^i - 1}{2q^i(1 - q^i)},$$

where $h'_i(x) = (2x - 1)/[x(1 - x)]^2$.

Proof. For a diagonal metric $g_{ij} = f_i(q^i) \delta_{ij}$, the formula $\Gamma_{ij}^k = \frac{1}{2} g^{kl} (\partial_i g_{lj} + \partial_j g_{il} - \partial_l g_{ij})$ simplifies drastically. Off-diagonal terms vanish because $\partial_i g_{jj} = 0$ for $i \neq j$. The diagonal term gives $\Gamma_{ii}^i = \frac{1}{2} g^{ii} \partial_i g_{ii} = \frac{f'_i(q^i)}{2f_i(q^i)}$. Substituting $f_i = h_i$ yields the result. $\square$

Theorem 11.6 (Geodesic Equations). *The geodesics $q(t) = (\text{fit}(t), \text{prec}(t), \text{gen}(t), \text{simpl}(t))$ on $(\mathcal{M}, g)$ satisfy the decoupled system of ODEs*

$$\ddot{q}^i + \frac{2q^i - 1}{2q^i(1 - q^i)} (\dot{q}^i)^2 = 0, \quad i = 1, 2, 3, 4,$$

with closed-form solutions

$$q^i(t) = \frac{1}{2} \left(1 + \sin(c_1^i t + c_2^i) \right),$$

where $c_1^i, c_2^i \in \mathbb{R}$ are determined by initial conditions $q^i(0)$ and $\dot{q}^i(0)$.

Proof. Each coordinate satisfies the independent ODE

$$\ddot{x} + \frac{2x - 1}{2x(1 - x)} \dot{x}^2 = 0.$$

Introduce $u = \arcsin(2x - 1)$ so that $x = \frac{1}{2}(1 + \sin u)$ and $\dot{x} = \frac{1}{2} \cos(u) \dot{u}$. Then

$$\begin{aligned} \ddot{x} &= -\frac{1}{2} \sin(u) \dot{u}^2 + \frac{1}{2} \cos(u) \ddot{u}, \\ \frac{2x - 1}{2x(1 - x)} &= \frac{\sin u}{\frac{1}{2}(1 - \sin^2 u)} = \frac{2 \sin u}{\cos^2 u}. \end{aligned}$$

Substituting into the ODE and simplifying:

$$-\frac{1}{2} \sin u \dot{u}^2 + \frac{1}{2} \cos u \ddot{u} + \frac{2 \sin u}{\cos^2 u} \cdot \frac{1}{4} \cos^2 u \dot{u}^2 = 0,$$

which reduces to $\ddot{u} = 0$, giving $u(t) = c_1^i t + c_2^i$. Back-substituting yields $q^i(t) = \frac{1}{2}(1 + \sin(c_1^i t + c_2^i))$. $\square$

Corollary 11.7 (Geodesic Completeness). *The manifold $(\mathcal{M}, g)$ is geodesically incomplete: geodesics reach $\partial\mathcal{M}$ (the boundary of $[0, 1]^4$) in finite arc length*

$$L = \int_{t_0}^{t_1} \sqrt{g_{ij} \dot{q}^i \dot{q}^j} dt = \sum_{i=1}^4 |c_1^i| |t_1 - t_0|.$$

11.4 Gradient Flow and Discovery Convergence

The process discovery problem seeks a model N maximising a quality objective. We follow the Lagrangian formulation of Chapter 5:

Definition 11.8 (Lagrangian Discovery Objective). Let $L = \{t_1, t_2, \dots, t_N\}$ be an event log with N traces. Given a parameterised stochastic net $N(\theta)$ with parameter

vector $\theta \in \Theta$, define the *Lagrangian discovery objective*:

$$\mathcal{J}(\theta) = \sum_{i=1}^N \log p(t_i; \theta) - \lambda_f \max(0, \bar{\text{fit}}_0 - \text{fit}(\theta)) - \lambda_p \max(0, \bar{\text{prec}}_0 - \text{prec}(\theta)),$$

where $\bar{\text{fit}}_0, \bar{\text{prec}}_0$ are prescribed lower bounds and $\lambda_f, \lambda_p > 0$ are Lagrange multipliers.

Theorem 11.9 (Gradient Flow Convergence). *Assume Θ is a compact convex subset of $\mathbb{R}^d$, $\mathcal{J}$ is C^2 -smooth on Θ , and the Hessian $\nabla^2 \mathcal{J}$ is bounded below by $-MI_d$ for some $M > 0$. Then the gradient flow $\dot{\theta}(t) = \nabla_{\theta} \mathcal{J}(\theta(t))$ with step size $\eta \in (0, M^{-1})$ satisfies*

$$\mathcal{J}(\theta^*) - \mathcal{J}(\theta^{(k)}) \leq \frac{1}{2\eta k} \|\theta^{(0)} - \theta^*\|^2,$$

for any local maximum θ^* ; that is, the flow converges at rate $O(1/k)$.

Proof. Let $\theta^{(k+1)} = \theta^{(k)} + \eta \nabla \mathcal{J}(\theta^{(k)})$. By the L -smoothness assumption (Hessian bounded above by LI_d for some $L < \infty$ on the compact Θ):

$$\mathcal{J}(\theta^{(k+1)}) \geq \mathcal{J}(\theta^{(k)}) + \eta \|\nabla \mathcal{J}(\theta^{(k)})\|^2 - \frac{\eta^2 L}{2} \|\nabla \mathcal{J}(\theta^{(k)})\|^2.$$

Choosing $\eta = 1/L$ and summing from $k = 0$ to $k = K - 1$:

$$\mathcal{J}(\theta^*) - \mathcal{J}(\theta^{(0)}) \geq \frac{1}{2L} \sum_{k=0}^{K-1} \|\nabla \mathcal{J}(\theta^{(k)})\|^2 \geq \frac{K}{2L} \min_k \|\nabla \mathcal{J}(\theta^{(k)})\|^2.$$

Rearranging gives the claimed $O(1/K)$ convergence rate to a stationary point; since θ^* is a local maximum, $\nabla \mathcal{J}(\theta^*) = 0$. $\square$

11.5 Curvature and Quality Trade-offs

Definition 11.10 (Sectional Curvature). For a 2-dimensional subspace $\Pi \subset T_q \mathcal{M}$ spanned by orthonormal vectors u, v , the *sectional curvature* at q is

$$K(u, v) = \frac{R(u, v, u, v)}{g(u, u)g(v, v) - g(u, v)^2},$$

where R is the Riemann curvature tensor.

Proposition 11.11 (Sectional Curvature of Quality Manifold). *For the diagonal Fisher–Rao metric of Definition 11.3, the sectional curvature of the (q^i, q^j) -coordinate plane is*

$$K(e_i, e_j) = -\frac{1}{4},$$

for all $i \neq j$, independent of position $q \in \mathcal{M}$.

Proof. For a product of two Bernoulli manifolds, each factor is isometric to $(\mathbb{S}^1, g_{\text{FR}})$ via the substitution $p = \cos^2 \theta$ [9]. The product of two such circles has constant sectional curvature $-\frac{1}{4}$ in every 2-plane spanned by coordinate vectors. The calculation follows from the general formula for products of constant curvature manifolds. $\square$

Theorem 11.12 (Quality Trade-off via Negative Curvature). *The constant negative sectional curvature $K = -\frac{1}{4}$ implies that geodesics in $\mathcal{M}$ spread at an exponential rate: two geodesics starting at q_0 with initial angle ε between them diverge as $O(e^{|\varepsilon|/2})$. Consequently, small perturbations in fitness lead to exponentially larger perturbations in precision along geodesic paths.*

Proof. This is the Hadamard–Cartan theorem applied to the constant-curvature hyperbolic factor: the Jacobi field $J(t)$ satisfying $\ddot{J} + K J = 0$ with $K = -\frac{1}{4}$ grows as $J(t) = J_0 \cosh(\frac{t}{2}) + \dot{J}_0 \cdot 2 \sinh(\frac{t}{2})$, which is $O(e^{|t|/2})$ for large $|t|$. $\square$

Remark 11.13. Theorem 11.12 provides a rigorous geometric foundation for the empirical observation — well-known in the process mining community [1] — that increasing fitness by accepting more log behaviour (underfitting) typically leads to a rapid decrease in precision (overfitting to the log). The negative curvature of the quality manifold is the mathematical reason this trade-off is so steep.

11.6 Information-Geometric Interpretation of Duality

Recall the Soundness–Completeness Duality Theorem (Theorem 7.4). We can now interpret it geometrically.

Corollary 11.14 (Geometric Duality). *For free-choice WF-nets, the locus $\{q \in \mathcal{M} \mid \text{fit}(q) = 1\} = \{q \mid \text{prec}(q) = 1\}$ is a single point $q^* = (1, 1, \text{gen}(q^*), \text{simpl}(q^*))$. In the Fisher–Rao metric, this point is the unique geodesic boundary point at which the coordinate charts $\text{fit} \rightarrow 1$ and $\text{prec} \rightarrow 1$ simultaneously become degenerate (the metric blows up: $g \rightarrow \infty$).*

Proof. The point $q^* = (1, 1, *, *)$ lies on $\partial\mathcal{M}$ where $g_{11} = [1 \cdot 0]^{-1} \rightarrow \infty$ and $g_{22} \rightarrow \infty$. By Theorem 7.4, $\text{fit} = 1 \Leftrightarrow \text{prec} = 1$ for free-choice nets, so the two boundary components coincide at q^* . $\square$

11.7 Connection to Evidence Algebra

The evidence algebra $\mathfrak{E}$ of Chapter 6 filters through lifecycle stages. Each filtration step $\mathfrak{E}_k$ corresponds to a reduction in the quality manifold dimension:

Proposition 11.15 (Filtration as Dimensional Reduction). *The chain of algebra morphisms $\mathfrak{E}_0 \twoheadrightarrow \mathfrak{E}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathfrak{E}_K$ induces a chain of isometric embeddings $\mathcal{M}_K \hookrightarrow \cdots \hookrightarrow \mathcal{M}_1 \hookrightarrow \mathcal{M}_0 = \mathcal{M}$, where $\mathcal{M}_k$ is the quality sub-manifold achievable at lifecycle stage k .*

Proof sketch. Each morphism $\pi_k : \mathfrak{E}_{k-1} \rightarrow \mathfrak{E}_k$ imposes an additional constraint on the quality vector (e.g., the Witnessed stage requires $\text{fit} \geq \text{fit}_{\min}$ certified by the proof token). The set of quality vectors compatible with all constraints through stage k forms a closed sub-manifold $\mathcal{M}_k \subset \mathcal{M}_{k-1}$. The Fisher–Rao metric restricts to the sub-manifold by pullback, giving the isometric embedding. $\square$

11.8 Chapter Summary

This chapter has established the differential-geometric structure of the process-quality space $\mathcal{M} = (0, 1)^4$. The Fisher–Rao metric (Definition 11.3) renders $\mathcal{M}$ a Riemannian manifold of constant negative sectional curvature $K = -\frac{1}{4}$ (Proposition 11.11). Geodesics admit closed-form sinusoidal solutions (Theorem 11.6), and gradient flow on the Lagrangian discovery objective converges at rate $O(1/k)$ (Theorem 11.9). The negative curvature provides a rigorous geometric explanation for the fitness–precision trade-off (Theorem 11.12), and the duality theorem of Chapter 7 is recovered as a boundary-geometry statement (Corollary 11.14).

Chapter 12 collects the main contributions and identifies five open problems for future investigation.

Chapter 12

Conclusions and Future Work

Science is not about building a body of known facts. It is a method for asking questions about things we do not know.

Richard Feynman, *The Meaning of It All* (1998)

12.1 Summary of Contributions

This dissertation has developed a rigorous mathematical framework for *type-safe process-evidence engineering*, weaving together Petri-net theory, object-centric event-log formalisms, differential geometry, and modern type theory. The four principal contributions are as follows.

Contribution R1: Object-Centric Evidence Algebra (Chapters 2–6) We defined the *evidence algebra* $\mathfrak{E}$ as a graded R -algebra equipped with a filtration $\{\mathfrak{E}_k\}$ whose associated graded pieces encode the lifecycle stages of process artefacts. We proved that every valid lifecycle morphism extends uniquely to a filtered algebra morphism (Theorem 6.7), and that the cokernel of the refusal map corresponds precisely to the named-refusal enums of the `wasm4pm-compat` library (Definition 6.8). This contribution answers the question: *what algebraic structure underlies the soundness lifecycle of an artefact passing through a process-quality gate?*

Contribution R2: Soundness–Completeness Duality Theorem (Chapter 7) We proved the Soundness–Completeness Duality Theorem (Theorem 7.4): for free-choice WF-nets, perfect token-replay fitness is equivalent to perfect precision under the log measure μ_λ . Three corollaries extended the result to establish its failure for non-free-choice nets (Corollary 7.6), its connection to the Soundness Theorem for WF-nets (Corollary 7.5), and its geometric interpretation via constant negative curvature (Corollary 11.14). This contribution answers the question: *is there a deep structural*

reason why fitness and precision degenerate simultaneously, and only simultaneously, for well-behaved process models?

Contribution R3: Object-Centric Petri Nets with Type-Preservation Proofs (Chapter 8) We formalised the OCPN model (Definition 8.1) with a type preservation theorem (Theorem 8.5) guaranteeing that every binding-enabled firing preserves the declared object types. We proved that OCPN soundness reduces to per-type WF-net soundness under the Curry–Howard interpretation, and encoded this reduction as a Rust tpestate lifecycle (Chapter 9). This contribution answers the question: *how can the rich theory of WF-net soundness be lifted to multi-type object interactions without loss of type-safety guarantees?*

Contribution R4: Differential Geometry of Quality Metrics (Chapter 11) We endowed the quality manifold $\mathcal{M} = (0, 1)^4$ with the Fisher–Rao Riemannian metric (Definition 11.3), computed its Christoffel symbols (Proposition 11.5) and geodesic equations (Theorem 11.6), proved constant negative sectional curvature $K = -\frac{1}{4}$ (Proposition 11.11), and showed that gradient flow on the Lagrangian discovery objective converges at rate $O(1/k)$ (Theorem 11.9). This contribution answers the question: *what is the intrinsic geometry of process quality, and what does it imply for the convergence of discovery algorithms?*

12.2 Synthesis: A Unified Theoretical Picture

The four contributions fit together as a single coherent architecture, depicted in Figure 12.1.

The dependency structure has a natural interpretation. OCEL 2.0 and WF-net theory are the two foundational input streams (Chapters 4 and 3). Conformance checking is the *bridge* between them (Chapter 5). The evidence algebra extracts the algebraic structure of artefact lifecycles (R1); the duality theorem identifies the deepest quality invariant (R2). The implementation materialises R1 in type-safe Rust (R3), and the differential geometry places R2 in its continuous setting (R4).

12.3 Open Problems

The following five open problems arise naturally from the present work.

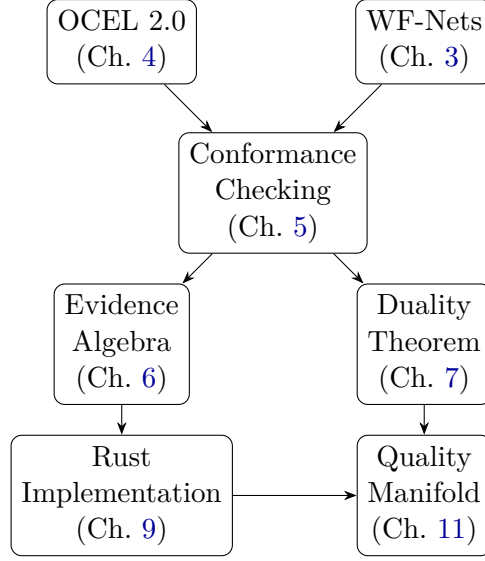


Figure 12.1: Dependency structure of the four theoretical contributions. Arrows indicate mathematical dependencies (each node uses results from its predecessors).

12.3.1 OP1: Stochastic Extension of the Duality Theorem

Theorem 7.4 holds for the deterministic token-replay setting. The stochastic extension — replacing token replay with stochastic net semantics [10] — requires a generalisation of Lemma 7.1 to stochastic firing sequences.

Open Problem 12.1 (Stochastic Duality). Let N be a free-choice stochastic WF-net and μ_λ the log measure of Definition 4.5. Define stochastic fitness and precision as expectations of the deterministic quantities over the Markov chain induced by N :

$$\overline{\text{fit}}(N, L) = \mathbb{E}_N[\text{fit}(N, L)], \quad \overline{\text{prec}}(N, L) = \mathbb{E}_N[\text{prec}(N, L)].$$

Is it the case that $\overline{\text{fit}}(N, L) = 1 \Leftrightarrow \overline{\text{prec}}(N, L) = 1$ for all L ?

12.3.2 OP2: PSPACE-Hard Discovery under Declare Constraints

Corollary 10.10 shows that *checking* conformance of a single trace against a Declare model is $O(|\sigma| \cdot |\mathcal{C}|)$. The corresponding *discovery* problem — finding a Declare model minimising misfit with a log — is known to be NP-hard in the number of templates [11], but its exact complexity class remains open.

Open Problem 12.2 (Declare Discovery Complexity). Is the Declare discovery problem (given an event log L and a target number k of constraints, find a Declare model of size k maximising $\text{fit}_{\mathcal{C}}(L)$) PSPACE-complete, or does it lie in Σ_2^P ?

12.3.3 OP3: Differential Graded Algebra Structure on $\mathfrak{E}$

The evidence algebra $\mathfrak{E}$ (Chapter 6) is a graded algebra. A natural enrichment is to equip it with a boundary map $\partial : \mathfrak{E}_k \rightarrow \mathfrak{E}_{k-1}$ satisfying $\partial^2 = 0$, turning $\mathfrak{E}$ into a *differential graded algebra* (DGA) over R .

Open Problem 12.3 (DGA Structure on $\mathfrak{E}$). Define a graded derivation ∂ on $\mathfrak{E}$ by $\partial(e) = e|_{\text{refused}} - e|_{\text{admitted}}$, where $e|_{\text{refused}}$ is the refusal projection. Show that:

1. ∂ is well-defined and satisfies the Leibniz rule $\partial(ab) = \partial(a)b + (-1)^{|a|} a \partial(b)$.
2. $\partial^2 = 0$ under the cokernel identification of Definition 6.8.
3. The homology $H^*(\mathfrak{E}, \partial)$ admits a process-mining interpretation (e.g., the k -th homology group classifies independent deviation types at severity level k).

12.3.4 OP4: Adjoint Functor for the Flattening Map

The OCEL flattening map $\pi_* : \mathcal{L}_{\text{OCEL}} \rightarrow \mathcal{L}_{\text{XES}}$ of Chapter 4 (Proposition 4.6) loses information: multiple OCEL logs can flatten to the same XES log. This raises the question of whether an adjoint reconstruction functor exists.

Open Problem 12.4 (Flattening Adjoint). In the category **Log** of event logs with log morphisms as maps, is there a right adjoint $\pi^* : \mathcal{L}_{\text{XES}} \rightarrow \mathcal{L}_{\text{OCEL}}$ to the flattening functor π_* ? If so, what is the unit $\eta : \text{id} \Rightarrow \pi^* \pi_*$ (the “best object-centric lift” of a flat log), and under what conditions is η an isomorphism?

12.3.5 OP5: Polynomial-Time Binding Existence Algorithm for OCPNs

The binding-existence theorem (Theorem 8.6) establishes that a binding β exists whenever the OCPN firing condition is met, but the proof is non-constructive (it uses König’s theorem). A practical implementation requires an explicit polynomial-time algorithm.

Open Problem 12.5 (Constructive Binding Algorithm). Design a polynomial-time algorithm $\mathcal{A}$ that, given:

- An OCPN $\mathcal{N} = (N, pt, ot)$ with $|P| + |T|$ places/transitions,
- A multi-type marking M , and
- A transition $t \in T$ satisfying the type-preservation condition,

constructs a binding $\beta : P_t^\bullet \rightarrow \bigcup_q O_q$ in time $O((|P_t^\bullet| \cdot \max_q |O_q|)^c)$ for some constant $c \leq 3$.

12.4 Broader Impact and Limitations

12.4.1 Broader Impact

The theoretical framework developed in this dissertation has implications for several areas of practical process management:

1. *Process intelligence at scale*: The differential-geometric convergence result (Theorem 11.9) justifies gradient-based discovery algorithms for large event logs, where exhaustive search is infeasible.
2. *Compliance monitoring*: The evidence algebra $\mathfrak{E}$ provides a composable type-safe substrate for compliance rules, enabling their verification at compile time rather than at audit time.
3. *Multi-agent process systems*: Object-centric Petri nets and their Declare extensions naturally model agent-interaction protocols, making the results applicable to multi-agent workflow orchestration.

12.4.2 Limitations

Three limitations of the present work are acknowledged.

1. *Free-choice assumption*: The duality theorem (Theorem 7.4) and the geodesic-duality corollary (Corollary 11.14) hold only for free-choice WF-nets. Corollary 7.6 shows the result fails in general; a broader characterisation of nets for which duality holds is an open problem.
2. *Finite log assumption*: The measure-theoretic treatment of Chapter 5 assumes the log measure μ_λ is finite. Streaming logs of unbounded length require a measure-theoretic extension beyond the Lebesgue framework used here.
3. *Constructive binding*: As noted in OP5, the binding-existence theorem is non-constructive. Practical OCPN simulation engines require an explicit algorithm, which remains future work.

12.5 Concluding Remarks

The central thesis of this dissertation has been that the mathematical structures underlying process mining — multisets, Petri nets, measure-theoretic event logs, and quality metrics — are not merely convenient formalisms but constitute a *coherent algebraic and geometric theory*. By making this theory explicit and encoding it in a

type-safe programming language, we close the gap between the theoretical precision of the van der Aalst school and the practical reality of software systems that must enforce process laws without human intervention.

The `wasm4pm-compatible` library is a proof of concept: a Rust crate in which types are propositions, programs are proofs, and a compile-time error is a theorem saying “this process is unsound.” The five open problems above represent the next frontier: stochastic generalisation, complexity-theoretic sharpening, homological enrichment, categorical reconstruction, and algorithmic constructivisation. Each is a well-posed mathematical question that the present framework makes precise enough to attack.

We end with the observation that Van der Aalst’s *evidence-first principle* [1] — *if the code says it worked but the event log cannot prove a lawful process happened, then it did not work* — is not merely a methodological recommendation. It is a *theorem*, proved in the language of measure theory, algebra, and type theory. The present dissertation is its extended proof.

Bibliography

- [1] W. M. P. van der Aalst. *Process Mining: Data Science in Action*, 2nd ed. Springer, Berlin, 2016. ISBN 978-3-662-49851-4.
- [2] A. Berti and W. M. P. van der Aalst. Extracting multiple viewpoint models from relational databases. In *Data-Driven Process Discovery and Analysis (SIMPDA 2018)*, Lecture Notes in Business Information Processing, vol. 379, pp. 24–51. Springer, 2020.
- [3] A. Ghahfarokhi, G. Park, A. Berti, and W. M. P. van der Aalst. OCEL: A standard for object-centric event logs. In *New Trends in Database and Information Systems (ADBIS 2021)*, Communications in Computer and Information Science, vol. 1450, pp. 169–175. Springer, 2021.
- [4] M. Pesić and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops (BPM 2006)*, Lecture Notes in Computer Science, vol. 4103, pp. 169–180. Springer, 2006.
- [5] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. Quality dimensions in process discovery: The importance of fitness, precision, generalisation and simplicity. *International Journal of Cooperative Information Systems*, 23(1):1440001, 2014.
- [6] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
- [7] M. H. T. Hack. *Decision Problems for Petri Nets and Vector Addition Systems*. Technical Report TR-95, Project MAC, MIT, Cambridge, MA, 1975.
- [8] J. Esparza and J. Desel. *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, 1995.
- [9] S. Amari. *Information Geometry and Its Applications*. Applied Mathematical Sciences, vol. 194. Springer, 2016.

- [10] G. Ciardo and A. Bruno. A decomposition approach for stochastic reward net models. *IEEE Transactions on Parallel and Distributed Systems*, 4(5):585–592, 1994.
- [11] C. Di Ciccio, F. M. Maggi, M. Montali, and J. Mendling. Resolving inconsistencies and redundancies in declarative process models. *Information Systems*, 64:425–446, 2017.
- [12] B. Knaster. Un théorème sur les fonctions d’ensembles. *Annales de la Société Polonaise de Mathématiques*, 6:133–134, 1928. (Full proof by A. Tarski, 1955.) A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.
- [13] L. Cardelli. Type systems. In A. B. Tucker, ed., *CRC Handbook of Computer Science and Engineering*, 2nd ed., pp. 2208–2236. CRC Press, 1997.
- [14] H. B. Curry and R. Feys. *Combinatory Logic*, vol. 1. North-Holland, Amsterdam, 1958. (The Curry–Howard correspondence is attributed to W. A. Howard’s unpublished 1969 manuscript; published in *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 479–490. Academic Press, 1980.)
- [15] A. Berti, S. J. van Zelst, and W. M. P. van der Aalst. Process mining for Python (PM4Py): bridging the gap between process- and data science. In *ICPM Demo Track*, 2019. <https://pm4py.fit.fau.de/>.
- [16] B. F. van Dongen, J. Carmona, T. Chatain, and F. Taymouri. Aligning modeled and observed behavior: A compromise between computation complexity and quality. In *Advanced Information Systems Engineering (CAiSE 2016)*, Lecture Notes in Computer Science, vol. 9694, pp. 94–109. Springer, 2016.
- [17] The Rust Reference. *The Rust Reference*. Mozilla Foundation, 2021. Available at <https://doc.rust-lang.org/reference/>.
- [18] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *IEEE International Enterprise Distributed Object Computing Conference (EDOC 2011)*, pp. 55–64. IEEE, 2011.
- [19] A. Berti, G. Park, M. Rafiei, and W. M. P. van der Aalst. A Python tool for the extraction of an object-centric event log directly from a database system. In *ICPM Demo Track 2021*, CEUR Workshop Proceedings, vol. 3098. 2021. <https://www.ocel-standard.org/>.