Laws of Agent Composition

Part II of *An Algebra of Agents*

Matthew Long
The YonedaAI Collaboration, YonedaAI Research Collective
Chicago, IL
matthew@yonedaai.com

September 2026

Abstract

We give an equational account of how agents compose. Fixing the agent object of Part I, an effectful Mealy coalgebra with typed termination and with failure as a terminal outcome, we define the composition signature: sequential composition with its unit and its diverging zero, immediate failure, delay, guarded, nondeterministic and probabilistic choice, the synchronous product, the first-halt product, recovery from failure, and guarded recursion with its derived loop. For each operator we prove that it is well defined on agents, that bisimilarity is a congruence for it, and which laws it satisfies at which relation. Agents and sequential composition form a category. The guarded fragment satisfies the axioms of Guarded Kleene Algebra with Tests up to bisimilarity whenever loop bodies consume a tick, and the axioms in which the diverging agent occurs on the right of a composition hold only under accepted traces. The unguarded choice is weaker in three separate ways: left distributivity fails up to bisimilarity, accepted-trace equality is not a congruence for sequential composition, and the algebra of accepted-trace sets has no multiplicative unit, because an accepted trace is read after a tick and never before the first, so an agent that halts at once is recorded as halting one tick later. The synchronous product is symmetric monoidal but violates the exchange law unless its factors halt in step. Guarded equations have unique solutions, which is why an unconditional retry loop has no meaning until a delay is inserted.

1 Introduction

Agent frameworks compose components by sequencing them, running them together, branching on a test, recovering from a failure and iterating until a goal is met. Process calculi, workflow engines and control software govern the same operations with equational theories that say which rearrangements preserve meaning. Agent systems implement the operations and have no such theory. This Part supplies one for the object fixed in Part I.

The series makes one claim, quoted here as in every Part.

> Part I fixes the object: discrete time, an effect monad T drawn from the list Id, $\mathcal{P}$, $\mathcal{D}$, and agents as effectful Mealy coalgebras with typed termination and failure as a terminal outcome. Parts II to IV introduce the operators under additional assumptions stated in each Part and prove, for each operator, that it is well-defined on agents, which relation it respects, and which laws hold at that relation: Part II for ;, $\oplus$, $\otimes$, $\triangleright$ and μ; Part III for Tr, channels, $\|$ and Gate_H; Part IV for Sup, Persist and Replay; Part V introduces no operator and interprets policies, plans, options, shields and controllers as terms of Parts I to IV, proving properties of their closed loops. For each

> of sixteen fields the series gives an explicit translation of one named mechanism into a term of the algebra together with an adequacy result for that translation, which is a soundness theorem, a one-way simulation, or a proved limitation; no field is claimed to reduce to the algebra.

This Part proves the operator half of that claim for sequential composition, the three choices, the synchronous product, the first-halt product, recovery and guarded recursion, under three added assumptions: loop bodies consume a tick, the arguments of the unguarded and probabilistic choices are running at their initial states, and the effect monad is commutative.

The guarded fragment of the signature is a model of Guarded Kleene Algebra with Tests up to bisimilarity, for every one of the three effect monads, with the single side condition that a loop body must consume a tick (Theorem 8.1). The corresponding statement for Kleene Algebra with Tests is weaker, and the reason is structural. A guarded choice is resolved at the initial state and costs nothing; an unguarded choice must be resolved by a transition, and a transition costs a tick.

That tick makes the unguarded choice weaker than the literature on nondeterministic process algebra would predict, and the weakness comes from typed termination rather than from nondeterminism. The halt outcome of Part I is a function of the state, so no single state can both hand on a value and go on to behave like an alternative branch, and an accepted trace is read after a tick and never before the first. The algebra of accepted-trace sets therefore has no multiplicative unit (Theorem 9.3); accepted-trace equality is not a congruence for sequential composition, so the quotient of all agents by that equality is not an algebra (Theorem 9.2); and the Kleene-algebra identities that survive do so on the fragment of agents that are running at their initial states (Theorem 9.4).

The synchronous product is a symmetric monoidal product up to bisimilarity for every commutative T (Theorem 10.1), and it fails the exchange law of concurrent Kleene algebra in both directions, even up to stuttering refinement, as soon as two branches have different durations (Theorem 10.3). The failure is not an artefact of the definition. A synchronous product halts when both factors have halted, so a fast factor waits, and waiting is observable. Part III recovers the exchange inclusion for the asynchronous interleaving built from channels; the operator that satisfies it is not this one.

2 Prior art

The equational study of sequential and branching composition begins with Kleene algebra. Kozen's axiomatisation [7] settles the equational theory of regular events: the axioms of an idempotent semiring together with the unfolding and induction axioms for the star are complete for the language model. Kleene algebra with tests [8] adds a Boolean subalgebra of tests, which makes conditionals and while loops derived operations and gives an equational account of while programs. Guarded Kleene algebra with tests [5] restricts union and iteration to their guarded forms, so that $\mathsf{if}\ b\ \mathsf{then}\ p\ \mathsf{else}\ q$ and $\mathsf{while}\ b\ \mathsf{do}\ p$ are primitive and the unrestricted sum and star are dropped; the equational theory then becomes decidable in nearly linear time rather than PSPACE-complete. The three systems form a scale, and the results below place the agent signature on it: the agent signature models the guarded system exactly and the unguarded one only in part.

Concurrent Kleene algebra [2] adds a second composition operator and relates it to the first by the exchange law $(p \parallel q)\,;(r \parallel s) \sqsubseteq (p\,;r) \parallel (q\,;s)$. The inequality is the algebraic content of interleaving: a concurrent composition of sequences is refined by the sequence of concurrent compositions. The synchronous product of this Part is not the operator that satisfies it, and Theorem 10.3 shows the failure is two-sided. Bergstra and Klop's treatment of synchronous cooperation [1] and Hoare's CSP [3] are the sources for the distinction this Part draws between a fully synchronous product and an interleaving, and Esterel [4] is the practical ancestor of the lockstep discipline: every component

consumes exactly one input per tick.

Recovery from failure is treated here as a handler-shaped construct rather than as an effect. Plotkin and Power's account of algebraic operations [13] and Plotkin and Pretnar's handlers [14] give the shape, but the frozen signature of Part I has no exception monad: failure is a terminal halt outcome present for every effect instance, so recovery is defined uniformly and not only for one monad. Orc [12] is the closest operational precedent, with its sequential, parallel and otherwise combinators over possibly failing services.

The unique-solution theorem for guarded recursive equations is Milner's, and the technique used below is bisimulation up to bisimilarity in the form given by Sangiorgi [15]. Guardedness in a process calculus means that every occurrence of the recursion variable sits under an action prefix. In the agent setting the corresponding measure is a tick: an unfolding must consume an input before it reaches the variable again. That substitution is what makes Theorem 7.3 say something about engineering practice rather than about syntax.

Typed termination breaks two properties that the calculi above take for granted. The algebra of accepted-trace sets has no multiplicative unit, and accepted-trace equality is not a congruence for sequential composition. In a process calculus, termination is an event recorded in the trace, so a composition can be read off the traces of its components; here it is a predicate on states, read after a tick, and the two facts fail. Both are proved below with two-state witnesses, and both are what force each law in this Part to name the relation at which it holds.

3 The agent object

An interface is a pair (I, Σ) of pointed sets with idle input $\varepsilon \in I$ and idle output $\tau \in \Sigma$. The effect monad T is Id, $\mathcal{P}$ (finite nonempty subsets) or $\mathcal{D}$ (finite-support distributions with rational weights). The failure alphabet E is a fixed finite set. All three are fixed in Part I, cited here and never redefined.

An agent A of sequential type $X \Rightarrow Y$ over the interface (I, Σ), with failure alphabet E, is a state space S_A together with three maps,

$$\mathsf{init}_A : X \to S_A, \qquad \mathsf{step}_A : S_A \times I \to T(\Sigma \times S_A), \qquad \mathsf{halt}_A : S_A \to Y + E + 1.$$

We write $\mathsf{halt}_A(s) = \mathsf{inl}(y)$ and say s is halted with value y, $\mathsf{halt}_A(s) = \mathsf{inm}(e)$ and say s is failed with reason e, and $\mathsf{halt}_A(s) = \mathsf{inr}(*)$ and say s is running. Part I imposes the stuttering rule as part of the definition: if s is not running then $\mathsf{step}_A(s, i) = \eta(\tau, s)$ for every $i \in I$. Halting and failing are evaluated between ticks and consume no tick.

A bisimulation between A and B of the same sequential type is a relation $R \subseteq S_A \times S_B$ such that $(\mathsf{init}_A(x), \mathsf{init}_B(x)) \in R$ for every $x \in X$; $\mathsf{halt}_A(s) = \mathsf{halt}_B(t)$ in $Y + E + 1$ for every $(s, t) \in R$; and $(\mathsf{step}_A(s, i), \mathsf{step}_B(t, i)) \in \mathrm{Rel}(T)(\Delta_\Sigma \times R)$ for every $(s, t) \in R$ and every $i \in I$, where $\mathrm{Rel}(T)$ is Barr relation lifting, which is equality for Id, the Egli-Milner lifting for $\mathcal{P}$ and the Larsen-Skou lifting for $\mathcal{D}$. Bisimilarity $\sim$ is the largest bisimulation. Two facts from Part I are used repeatedly and are recorded here. Relation lifting commutes with the monad structure, so if u and v are $\mathrm{Rel}(T)(R)$-related and f, g are functions that send R-related arguments to R'-related results, then $T(f)(u)$ and $T(g)(v)$ are $\mathrm{Rel}(T)(R')$-related. And the lifting of the graph of a function f is the graph of $T(f)$ (Part I, Lemma 2.8), so a coalgebra homomorphism is a bisimulation (Part I, Lemma 4.3). Table 1 lists the imports with the numbers they carry in Part I.

The trace semantics of Part I is $[\![A]\!] : X \times I^* \to T(\Sigma^* \times (Y + E + 1))$, taking a start value and an input word of length n to the effectful collection of pairs consisting of the emitted output word of length n and the outcome after n ticks. The empty input word is included, and $[\![A]\!](x, \epsilon)$ records

Table 1: Objects imported from Part I. Each is cited by the label and number its owner fixes; this Part defines none of them.

Object	Label in Part I	Number
Interface (I, Σ) and the product interface	`def:agent-object:interface`	Definition 2.1
Effect monad T and the instances	`def:agent-object:effect-monad`	Definition 2.2
Relation lifting $\mathrm{Rel}(T)$ and predicate lifting	`def:agent-object:lifting`	Definition 2.4
Compatibility of $\mathrm{Rel}(T)$ with bind and graphs	`lem:agent-object:lifting-bind`	Lemma 2.8
Agent $A : X \Rightarrow Y$ and the stuttering rule	`def:agent-object:agent`	Definition 2.10
Single transition $s \to^{i/\sigma} s'$	`def:agent-object:transition`	Definition 2.11
Decorated and undecorated semantics	`def:agent-object:trace`	Definition 3.2
Trace equivalence $\simeq_{\mathrm{tr}}$	`def:agent-object:trace-equiv`	Definition 3.3
Splitting of runs	`lem:agent-object:prefix`	Lemma 3.4
Accepted traces and $\simeq_{\mathrm{acc}}$	`def:agent-object:accepted-trace`	Definition 3.5
Bisimulation and bisimilarity $\sim$	`def:agent-object:bisim`	Definition 4.1
Coinduction: $\sim$ is the largest bisimulation	`lem:agent-object:coinduction`	Lemma 4.2
Homomorphisms are bisimulations	`lem:agent-object:hom`	Lemma 4.3
Bisimilarity implies trace equivalence	`thm:agent-object:bisim-implies-trace`	Theorem 4.4
Stutter closure	`def:agent-object:stutter`	Definition 5.3
Refinement $\sqsubseteq$	`def:agent-object:refinement`	Definition 5.4

$\mathsf{halt}_A(\mathsf{init}_A(x))$. An accepted trace is one whose outcome is $\mathsf{inl}(y)$; $\mathsf{acc}(A)$ denotes the family of accepted traces and $\simeq_{\mathrm{acc}}$ their equality. Refinement $\sqsubseteq$ is inclusion of stutter-closed trace sets, where the stutter closure inserts and deletes ticks carrying the idle pair (ε, τ) at running positions.

Remark 3.1. Two conventions of Part I do work in every proof below. Halting is silent, so a composite may hand a value from one component to the next between two ticks without emitting anything; and non-running states stutter, so a halted or failed component continues to consume inputs and emit τ while its neighbours run. The first convention is what makes sequential composition associative on the nose; the second is what makes the synchronous product wait.

4 The signature

Fix an interface (I, Σ) and a failure alphabet E. Every construction below produces an agent over that interface, or, for the synchronous product, over a product interface built from two of them. Each is given as an explicit coalgebra, because the laws are proved by exhibiting relations between state spaces and a state space is the only place such a relation can live.

4.1 Units, constants and delay

Definition 4.1 (identity)**.** $\mathsf{skip}_X : X \Rightarrow X$ has state space X, $\mathsf{init}(x) = x$, $\mathsf{step}(x, i) = \eta(\tau, x)$ and $\mathsf{halt}(x) = \mathsf{inl}(x)$. It halts at its initial state with the value it was started at. For a set Z and an element $z \in Z$, the constant agent $\iota_z : X \Rightarrow Z$ is the same coalgebra with $\mathsf{halt}(x) = \mathsf{inl}(z)$.

Definition 4.2 (divergence)**.** $\mathbf{0}_{X,Y} : X \Rightarrow Y$ has a single state $*$, $\mathsf{init}(x) = *$, $\mathsf{step}(*, i) = \eta(\tau, *)$ and $\mathsf{halt}(*) = \mathsf{inr}(*)$. It runs forever and emits the idle output. It is not a silent failure and not an empty choice.

Definition 4.3 (immediate failure)**.** For $e \in E$, $\mathsf{fail}_e^{X,Y} : X \Rightarrow Y$ has a single state $*$ with $\mathsf{step}(*, i) = \eta(\tau, *)$ and $\mathsf{halt}(*) = \mathsf{inm}(e)$. It fails at its initial state and consumes no tick.

Definition 4.4 (delay)**.** For $d \geq 0$, $\mathsf{delay}_d^X : X \Rightarrow X$ has state space $X \times \{0, \ldots, d\}$, $\mathsf{init}(x) = (x, d)$, $\mathsf{step}((x, k), i) = \eta(\tau, (x, \max(k-1, 0)))$, $\mathsf{halt}((x, 0)) = \mathsf{inl}(x)$ and $\mathsf{halt}((x, k)) = \mathsf{inr}(*)$ for $k > 0$. It consumes exactly d ticks and hands on the value it was started at. $\mathsf{delay}_0 = \mathsf{skip}$.

Definition 4.5 (engagement)**.** An agent $A : X \Rightarrow Y$ is *engaged* if $\mathsf{init}_A(x)$ is running for every $x \in X$.

$\mathbf{0}$ and delay_d for $d \geq 1$ are engaged; skip and fail_e are not. Engagement is exactly the condition under which an agent still has a first transition to contribute to a choice, and it is the hypothesis that separates the laws of the guarded choice from the laws of the unguarded one.

4.2 Sequential composition

Sequential composition runs A until it halts, then starts B at the value A halted with. Because halting consumes no tick, the replacement of the halted state of A by the initial state of B happens between two ticks, and B takes its first transition on the input of the following tick. A failure of A is not caught: it propagates.

Definition 4.6 (sequential composition)**.** Let $A : X \Rightarrow Y$ and $B : Y \Rightarrow Z$. Write $S_A^\circ = \{s \in S_A : s \text{ is not halted}\}$ and define the *handoff map* $h_{A,B} : S_A \to S_A^\circ + S_B$ by

$$h_{A,B}(s) \;=\; \begin{cases} \mathsf{inr}(\mathsf{init}_B(y)) & \text{if } \mathsf{halt}_A(s) = \mathsf{inl}(y), \\ \mathsf{inl}(s) & \text{otherwise.} \end{cases}$$

Then $A\,;B : X \Rightarrow Z$ has state space $S_A^\circ + S_B$, initial state $h_{A,B}(\mathsf{init}_A(x))$, and

$$\mathsf{step}(\mathsf{inl}(s), i) = T(\mathrm{id}_\Sigma \times h_{A,B})(\mathsf{step}_A(s, i)), \qquad \mathsf{step}(\mathsf{inr}(t), i) = T(\mathrm{id}_\Sigma \times \mathsf{inr})(\mathsf{step}_B(t, i)),$$

$$\mathsf{halt}(\mathsf{inl}(s)) = \begin{cases} \mathsf{inm}(e) & \text{if } \mathsf{halt}_A(s) = \mathsf{inm}(e) \\ \mathsf{inr}(*) & \text{otherwise,} \end{cases} \qquad \mathsf{halt}(\mathsf{inr}(t)) = \mathsf{halt}_B(t).$$

Lemma 4.7. *$A\,;B$ is an agent: its halt map is well defined on $S_A^\circ + S_B$, its step map lands in $T(\Sigma \times (S_A^\circ + S_B))$, and it satisfies the stuttering rule.*

Proof. The handoff map sends every halted state of A into S_B, so no state of the form $\mathsf{inl}(s)$ with s halted occurs in $S_A^\circ + S_B$, and the two clauses of halt are exhaustive. The step map is the image of step_A or step_B under a function into $S_A^\circ + S_B$, so it lands where required. For the stuttering rule, let $\mathsf{inl}(s)$ be failed, so $\mathsf{halt}_A(s) = \mathsf{inm}(e)$; then $\mathsf{step}_A(s, i) = \eta(\tau, s)$ by the rule for A and $h_{A,B}(s) = \mathsf{inl}(s)$, so $\mathsf{step}(\mathsf{inl}(s), i) = \eta(\tau, \mathsf{inl}(s))$. Let $\mathsf{inr}(t)$ be halted or failed; then $\mathsf{step}_B(t, i) = \eta(\tau, t)$ and the claim follows. □

Definition 4.8 (the category of agents)**.** $\mathsf{Ag}(I, \Sigma)$ has as objects the value sets and as morphisms $X \to Y$ the agents $X \Rightarrow Y$ modulo bisimilarity, with identity $[\mathsf{skip}_X]$ and composition $[A] \cdot [B] = [A\,;B]$.

That this is a category is Theorem 6.1; that the composition is well defined on bisimilarity classes is the sequential case of Theorem 5.1. The failure alphabet E is fixed alongside the interface, and we suppress it from the notation.

4.3 Choice

Three choices are needed and they differ in when the branch is selected. A guarded choice inspects the start value and selects a branch before the first tick. An unguarded choice has no value to inspect, so it must select its branch by taking a transition, and the selection therefore occupies a tick. Every asymmetry between the guarded laws and the unguarded ones traces back to this.

Definition 4.9 (tests)**.** A *test* on X is a function $b : X \to 2$. Tests form a Boolean algebra B_X under pointwise conjunction, disjunction and complement, with top 1 and bottom 0.

Definition 4.10 (guarded choice)**.** Let $A, B : X \Rightarrow Y$ and let b be a test on X. Then $A \oplus_b B : X \Rightarrow Y$ has state space $S_A + S_B$, initial state $\mathsf{inl}(\mathsf{init}_A(x))$ if $b(x) = 1$ and $\mathsf{inr}(\mathsf{init}_B(x))$ otherwise, and step and halt maps inherited through the two injections.

Definition 4.11 (test agent)**.** For a test b on X, put $[b] := \mathsf{skip}_X \oplus_b \mathbf{0}_{X,X}$. It halts at once with x when $b(x) = 1$ and diverges silently when $b(x) = 0$.

Definition 4.12 (nondeterministic choice)**.** Let $T = \mathcal{P}$ and $A, B : X \Rightarrow Y$. Then $A \oplus B : X \Rightarrow Y$ has state space $X + S_A + S_B$, initial state $\mathsf{inl}(x)$, and

$$\mathsf{step}(\mathsf{inl}(x), i) = (\mathrm{id} \times \mathsf{inm})\,[\mathsf{step}_A(\mathsf{init}_A(x), i)] \cup (\mathrm{id} \times \mathsf{inr})\,[\mathsf{step}_B(\mathsf{init}_B(x), i)]\,,$$
$$\mathsf{step}(\mathsf{inm}(s), i) = (\mathrm{id} \times \mathsf{inm})\,[\mathsf{step}_A(s, i)]\,, \qquad \mathsf{step}(\mathsf{inr}(t), i) = (\mathrm{id} \times \mathsf{inr})\,[\mathsf{step}_B(t, i)]\,,$$

with $\mathsf{halt}(\mathsf{inl}(x)) = \mathsf{inr}(*)$, $\mathsf{halt}(\mathsf{inm}(s)) = \mathsf{halt}_A(s)$ and $\mathsf{halt}(\mathsf{inr}(t)) = \mathsf{halt}_B(t)$. Here inl, inm and inr are the three injections into $X + S_A + S_B$.

Definition 4.13 (probabilistic choice)**.** Let $T = \mathcal{D}$, let $p \in [0, 1]$ be rational, and let $A, B : X \Rightarrow Y$. Then $A \oplus_p B$ is defined exactly as $A \oplus B$ except that the first step is the convex combination

$$\mathsf{step}(\mathsf{inl}(x), i) \;=\; p \cdot \mathcal{D}(\mathrm{id} \times \mathsf{inm})(\mathsf{step}_A(\mathsf{init}_A(x), i)) \;+\; (1 - p) \cdot \mathcal{D}(\mathrm{id} \times \mathsf{inr})(\mathsf{step}_B(\mathsf{init}_B(x), i)).$$

The initial state of an unguarded choice is running, which is forced: the halt map is a function into $Y + E + 1$ and a state that is not running stutters forever, so no single state can both accept a value and go on to behave like an alternative branch. A branch that is not engaged therefore reports its halt one tick later inside the choice than it would alone. That one tick is invisible to the accepted traces of Part I, which are read after ticks, and visible to bisimilarity, which is why the engagement hypothesis appears in the bisimilarity laws for $\oplus$ and not in the accepted-trace laws.

4.4 Products

Definition 4.14 (synchronous product)**.** Let $A_j : X_j \Rightarrow Y_j$ over (I_j, Σ_j) with failure alphabet E_j, for $j = 1, 2$. Then $A_1 \otimes A_2 : X_1 \times X_2 \Rightarrow Y_1 \times Y_2$ over the product interface $(I_1 \times I_2, \Sigma_1 \times \Sigma_2)$ with idle symbols $(\varepsilon, \varepsilon)$ and (τ, τ) has state space $S_1 \times S_2$, initial state $(\mathsf{init}_1(x_1), \mathsf{init}_2(x_2))$, and

$$\mathsf{step}((s_1, s_2), (i_1, i_2)) \;=\; T(\gamma)\big(\mathsf{dst}(\mathsf{step}_1(s_1, i_1), \mathsf{step}_2(s_2, i_2))\big),$$

where $\mathsf{dst} : T(U) \times T(V) \to T(U \times V)$ is the double strength of the commutative monad T and γ rearranges $(\Sigma_1 \times S_1) \times (\Sigma_2 \times S_2)$ into $(\Sigma_1 \times \Sigma_2) \times (S_1 \times S_2)$. The failure alphabet is $E_1 + E_2 + E_1 \times E_2$ and

$$\mathsf{halt}(s_1, s_2) = \begin{cases} \mathsf{inm}(e_1, e_2) & \text{if } s_1, s_2 \text{ are failed with } e_1, e_2, \\ \mathsf{inm}(e_j) & \text{if exactly } s_j \text{ is failed}, \\ \mathsf{inl}(y_1, y_2) & \text{if } s_1, s_2 \text{ are halted with } y_1, y_2, \\ \mathsf{inr}(*) & \text{otherwise.} \end{cases}$$

Definition 4.15 (unit of the product)**.** The unit interface is $(1, 1)$ with both alphabets the one-point set, and the unit agent is $\mathsf{skip}_1 : 1 \Rightarrow 1$.

Definition 4.16 (first-halt product)**.** With the data of Theorem 4.14, $\mathsf{race}(A_1, A_2)$ has the same state space, initial state and step map, sequential type $X_1 \times X_2 \Rightarrow Y_1 + Y_2 + Y_1 \times Y_2$ and halt map

$$\mathsf{halt}(s_1, s_2) = \begin{cases} \mathsf{inl}(y_1, y_2) & \text{if } s_1, s_2 \text{ are halted with } y_1, y_2, \\ \mathsf{inl}(\iota_j(y_j)) & \text{if exactly } s_j \text{ is halted with } y_j, \\ \mathsf{inm}(e_1, e_2) & \text{if } s_1, s_2 \text{ are failed with } e_1, e_2, \\ \mathsf{inr}(*) & \text{otherwise.} \end{cases}$$

The product halts when both factors have halted; the race halts as soon as one has, and fails only when both have failed. The two therefore differ only in the halt map, which is why every law that concerns only outputs holds for both. Recording the outcome of a simultaneous halt as a pair, rather than breaking the tie to the left, is what makes the race commutative up to the evident symmetry rather than up to a hypothesis about tie-breaking.

4.5 Recovery

Definition 4.17 (recovery)**.** Let $A : X \Rightarrow Y$ have failure alphabet E and let $B : E \Rightarrow Y$. Write $S_A^\dagger = \{s \in S_A : s \text{ is not failed}\}$ and define $r_{A,B} : S_A \to S_A^\dagger + S_B$ by $r_{A,B}(s) = \mathsf{inr}(\mathsf{init}_B(e))$ when $\mathsf{halt}_A(s) = \mathsf{inm}(e)$ and $r_{A,B}(s) = \mathsf{inl}(s)$ otherwise. Then $A \rhd B : X \Rightarrow Y$ has state space $S_A^\dagger + S_B$, initial state $r_{A,B}(\mathsf{init}_A(x))$,

$$\mathsf{step}(\mathsf{inl}(s), i) = T(\mathrm{id} \times r_{A,B})(\mathsf{step}_A(s, i)), \qquad \mathsf{step}(\mathsf{inr}(t), i) = T(\mathrm{id} \times \mathsf{inr})(\mathsf{step}_B(t, i)),$$

$\mathsf{halt}(\mathsf{inl}(s)) = \mathsf{halt}_A(s)$ (which is halted or running), and $\mathsf{halt}(\mathsf{inr}(t)) = \mathsf{halt}_B(t)$. Its failure alphabet is that of B.

Recovery is the mirror image of sequential composition: the same silent replacement of a terminal state by an initial state, triggered by the other terminal outcome, and passing the failure reason rather than the halting value. The handler receives the reason, so $\mathsf{fail}_e \rhd B$ starts B at e.

4.6 Recursion

Recursion needs a syntax, because guardedness is a property of a term and not of an agent. Fix a set of agent constants and a variable X.

Definition 4.18 (contexts)**.** A *context* is generated by

$$F ::= \mathsf{X} \mid A \mid F\,;F \mid F \oplus_b F \mid F \oplus F \mid F \oplus_p F \mid F \otimes F \mid \mathsf{race}(F, F) \mid F \rhd F \mid \mu\mathsf{Y}.F,$$

where A ranges over agent constants of the appropriate types and the type discipline of Section 4 is respected at every node. $F(C)$ denotes the result of substituting the agent C for every free occurrence of X.

A *runtime term* of F is a copy of F in which every constant has been replaced by a state of that constant and every occurrence of X is marked either pending or already unfolded. Three rewriting steps are silent, in the sense that they change a runtime term without consuming a tick: the handoff of Theorem 4.6 when the left component is halted, the recovery of Theorem 4.17 when the left component is failed, and the unfolding of a pending X into a fresh runtime copy of F. Write $\rightsquigarrow$ for the union of the three and call a runtime term *settled* when it is $\rightsquigarrow$-normal.

Definition 4.19 (guardedness)**.** A context F is *guarded* if $\rightsquigarrow$ terminates from every runtime term of F.

Definition 4.20 (guarded recursion)**.** For a guarded context F with X of type $X \Rightarrow Y$, $\mu\mathsf{X}.F(\mathsf{X}) : X \Rightarrow Y$ is the agent whose states are the settled runtime terms of F, whose initial state is the settling of the runtime term of F with every constant at its initial state for x and every X pending, whose step map is the operator-by-operator step of Section 4 followed by settling, and whose halt map is the operator-by-operator halt map.

Lemma 4.21. *For guarded F, Theorem 4.20 defines an agent, and settling is confluent, so the result does not depend on the order in which the silent steps are taken.*

Proof. Termination is the definition of guardedness. Confluence holds because the three silent steps act at disjoint positions of a runtime term and none of them creates or destroys a redex at another position: a handoff replaces a halted left component by an initial right component, a recovery replaces a failed left component by an initial right component, and an unfolding replaces a pending variable by a runtime copy. Positions are therefore rewritten independently and the normal form is unique. The halt map is well defined on settled terms because a settled term has no halted left component of a sequential node, no failed left component of a recovery node, and no pending variable. □

Definition 4.22 (loop)**.** For $A : X \Rightarrow X$ and a test b on X, $\mathsf{while}_b(A) := \mu\mathsf{X}.((A\,;\mathsf{X}) \oplus_b \mathsf{skip}_X)$, defined whenever the context is guarded.

The syntactic criterion below is the one used in practice. It is sufficient and not necessary, which is the usual trade for decidability.

Lemma 4.23 (syntactic guardedness)**.** *Define $\rho(F) \in \{0,1\}$ by $\rho(\mathsf{X}) = 1$; $\rho(A) = 0$ for a constant; $\rho(F\,;G) = \rho(F)$ if F is an engaged constant and $\max(\rho(F),\rho(G))$ otherwise; $\rho(F \rhd G) = \max(\rho(F),\rho(G))$; and $\rho(F \star G) = \max(\rho(F),\rho(G))$ for $\star$ any of $\oplus_b$, $\oplus$, $\oplus_p$, $\otimes$ and race. If $\rho(F) = 0$ then F is guarded.*

Proof. $\rho(F) = 0$ means that no silent path through F reaches an occurrence of X: a handoff can only be taken past a component that halts before its first tick, a recovery only past one that fails before its first tick, and the clauses of ρ mark each of those cases. Every $\rightsquigarrow$-chain therefore has length bounded by the number of nodes of F times the number of nested unfoldings needed to reach a constant that must consume a tick, which is finite. □

In particular $\mathsf{delay}_d;\mathsf{X}$ is guarded for $d \geq 1$ and $\mathsf{skip};\mathsf{X}$ is not, and neither is X under an unguarded choice: the choice of Theorem 4.12 selects a branch by taking a transition, but it initialises the selected branch inside that same transition, so it does not by itself supply the tick.

5 Congruence

The operators descend to bisimilarity classes because relation lifting commutes with the action of a function on T and is closed under the union of $\mathcal{P}$-values and under convex combination of $\mathcal{D}$-values with equal weights. Part I proves both properties, and one construction per operator turns them into the congruence theorem.

Theorem 5.1 (congruence)**.** *Let $A \sim A'$ and $B \sim B'$ at the appropriate types. Then*

$$A\,;B \sim A'\,;B', \quad A \oplus_b B \sim A' \oplus_b B', \quad A \oplus B \sim A' \oplus B', \quad A \oplus_p B \sim A' \oplus_p B',$$

$$A \otimes B \sim A' \otimes B', \quad \mathsf{race}(A,B) \sim \mathsf{race}(A',B'), \quad A \rhd B \sim A' \rhd B', \quad \mathsf{delay}_d\,;A \sim \mathsf{delay}_d\,;A',$$

and for a guarded context F whose constants are replaced by bisimilar constants, $\mu\mathsf{X}.F(\mathsf{X}) \sim \mu\mathsf{X}.F'(\mathsf{X})$. The choices $\oplus$ and $\oplus_p$ require $T = \mathcal{P}$ and $T = \mathcal{D}$ respectively; the product requires T commutative.

Proof. Fix bisimulations $R \subseteq S_A \times S_{A'}$ and $Q \subseteq S_B \times S_{B'}$.

Sequential composition. Put $R\,;Q = \{(\mathsf{inl}(s), \mathsf{inl}(s')) : (s,s') \in R\} \cup \{(\mathsf{inr}(t), \mathsf{inr}(t')) : (t,t') \in Q\}$, restricted to the state spaces of Theorem 4.6. Initial states are related because $(\mathsf{init}_A(x), \mathsf{init}_{A'}(x)) \in R$ and, if both are halted with the same value y (they are, since R preserves halt), then both handoff maps produce inr of initial states of B and B' at y, which are Q-related. The halt maps agree by inspection of the two clauses. For the step maps, the handoff maps $h_{A,B}$ and $h_{A',B'}$ send R-related arguments to $(R\,;Q)$-related results, again because R preserves halt and Q relates initial states; relation lifting then carries the $\mathrm{Rel}(T)(\Delta_\Sigma \times R)$-relatedness of $\mathsf{step}_A(s,i)$ and $\mathsf{step}_{A'}(s',i)$ to $\mathrm{Rel}(T)(\Delta_\Sigma \times (R\,;Q))$- relatedness of their images.

Guarded choice. Take $\mathsf{inl}(R) \cup \mathsf{inr}(Q)$. The initial states are related because b selects the same branch on both sides, and the step and halt maps are inherited.

Nondeterministic choice. Take $\Delta_X \cup \mathsf{inm}(R) \cup \mathsf{inr}(Q)$ on $X + S_A + S_B$ and $X + S_{A'} + S_{B'}$. Both merged initial states are running, so the halt condition holds there. For the step, the two first-step sets are unions of the images of $\mathsf{step}_A(\mathsf{init}_A(x), i)$ and $\mathsf{step}_B(\mathsf{init}_B(x), i)$, and Egli-Milner lifting is closed under union of related sets. Elsewhere the relation is inherited.

Probabilistic choice. The same relation. Larsen-Skou lifting is closed under convex combination with matching weights, and the weights p and $1-p$ are the same on both sides.

Product. Take $R \times Q$. The halt maps agree because each clause of Theorem 4.14 depends only on the pair of outcomes, which R and Q preserve. For the step, the double strength of a commutative monad preserves relation lifting: if u, u' are $\mathrm{Rel}(T)(R_1)$-related and v, v' are $\mathrm{Rel}(T)(R_2)$-related then $\mathsf{dst}(u,v)$ and $\mathsf{dst}(u',v')$ are $\mathrm{Rel}(T)(R_1 \times R_2)$-related. The same relation works for the race, whose halt map again depends only on the pair of outcomes.

Recovery. Take $\mathsf{inl}(R) \cup \mathsf{inr}(Q)$ as for sequential composition, with the recovery map $r_{A,B}$ in place of the handoff map; R preserves failure reasons, so the two recovery maps agree on which reason is passed to B and B'.

Delay. delay_d has no constants to vary, and $\mathsf{delay}_d\,;A \sim \mathsf{delay}_d\,;A'$ is the sequential case.

Recursion. Relate two settled runtime terms when they have the same shape and their constants are pointwise related by R or Q. Settling preserves the relation: a handoff is taken on both sides simultaneously because related states have equal outcomes, and likewise for a recovery and for an unfolding, which is driven by the shape alone. The step of a settled term is the operator-by-operator step, and the cases above show that each operator preserves relatedness; the subsequent settling preserves it by the same argument. The relation is therefore a bisimulation. □

The hypothesis that halting and failing are state predicates is used in every case: it is what allows a bisimulation to be a relation on states rather than a relation on states paired with a history. The race uses it twice, once for each factor.

6 The category of agents

Theorem 6.1 (agents form a category)**.** *For every interface (I,Σ), every failure alphabet E and every $T \in \{\mathrm{Id}, \mathcal{P}, \mathcal{D}\}$, the data of Theorem 4.8 form a category. That is, $(A\,;B)\,;C \sim A\,;(B\,;C)$ for all $A : X \Rightarrow Y$, $B : Y \Rightarrow Z$, $C : Z \Rightarrow W$, and $\mathsf{skip}_X\,;A \sim A \sim A\,;\mathsf{skip}_Y$ for all $A : X \Rightarrow Y$.*

Proof. Composition is well defined on classes by Theorem 5.1.

Left unit. $\mathsf{skip}_X\,;A$ has state space $\mathsf{skip}_X^\circ + S_A$, and skip_X° is empty because every state of skip_X is halted. Its initial state is $\mathsf{inr}(\mathsf{init}_A(x))$, and its step and halt maps are those of A transported along inr. The graph of inr is therefore a bisimulation.

Right unit. $A\,;\mathsf{skip}_Y$ has state space $S_A^\circ + Y$. Put $R = \{(\mathsf{inl}(s), s) : s \in S_A^\circ\} \cup \{(\mathsf{inr}(y), s) : \mathsf{halt}_A(s) = \mathsf{inl}(y)\}$. Initial states are related, since $h_{A,\mathsf{skip}}(\mathsf{init}_A(x))$ is $\mathsf{inl}(\mathsf{init}_A(x))$ or $\mathsf{inr}(y)$ according as $\mathsf{init}_A(x)$ is halted, and both cases lie in R. Halt maps agree: on the first family both sides are failed with the same reason or running, and on the second both are halted with y. For steps, on the second family the left side stutters at $\mathsf{inr}(y)$ and the right side stutters at s by the stuttering rule, and the pair stays in R. On the first family the left step is the image of $\mathsf{step}_A(s, i)$ under $\mathrm{id} \times h_{A,\mathsf{skip}}$ and $h_{A,\mathsf{skip}}$ is a function whose graph, read backwards, is exactly R; the lifting of the graph of a function relates u to $T(f)(u)$, which is the claim.

Associativity. Write $N_1 = (S_A^\circ + S_B^\circ) + S_C$ for the state space of $(A;B);C$ and $N_2 = S_A^\circ + (S_B^\circ + S_C)$ for that of $A\,;(B\,;C)$; the intermediate space $(A\,;B)^\circ$ is $S_A^\circ + S_B^\circ$, because a state $\mathsf{inl}(s)$ of $A\,;B$ is never halted and a state $\mathsf{inr}(t)$ is halted exactly when t is. The canonical bijection $\alpha : N_1 \to N_2$ is a coalgebra isomorphism, and it suffices to check that it commutes with the two handoff maps. On S_A, the outer handoff of $(A\,;B)\,;C$ is $h' \circ h_{A,B}$ where h' is the handoff of $A\,;B$ into C; a halted s with value y is sent by $h_{A,B}$ to $\mathsf{inr}(\mathsf{init}_B(y))$, and then h' sends it to $\mathsf{inr}(\mathsf{init}_C(z))$ if $\mathsf{init}_B(y)$ is halted with z and leaves it otherwise. On the same s, the handoff of $A\,;(B\,;C)$ is $s \mapsto \mathsf{inr}(\mathsf{init}_{B;C}(y)) = \mathsf{inr}(h_{B,C}(\mathsf{init}_B(y)))$, which is $\mathsf{inr}(\mathsf{inr}(\mathsf{init}_C(z)))$ in the first case and $\mathsf{inr}(\mathsf{inl}(\mathsf{init}_B(y)))$ in the second. The bijection α matches the two cases. The remaining components of the step and halt maps are inherited identically on both sides. □

Remark 6.2. The cascade in the associativity proof is the reason the theorem is not immediate. A single halting value can trigger two handoffs before the next tick, and the two bracketings take them in different orders. They agree because each handoff advances along the finite list of components. The same cascade is unbounded for an unguarded recursion, which is exactly what Theorem 4.19 forbids.

Remark 6.3. $\mathsf{Ag}(I, \Sigma)$ is not the Kleisli category of T, and the resemblance is superficial. Its morphisms are coalgebras rather than maps $X \to T(Y)$, and its composition is a state-space construction rather than a bind. The connection is one level down: the step map of an agent is a Kleisli arrow $S \times I \to T(\Sigma \times S)$, and the monad structure of T appears in the proofs through relation lifting rather than through composition of morphisms.

7 Recursion and guardedness

Theorem 7.1 (unique solutions)**.** *Let F be a guarded context. Then $\mu\mathsf{X}.F(\mathsf{X}) \sim F(\mu\mathsf{X}.F(\mathsf{X}))$, and if $C \sim F(C)$ for an agent C of the same type then $C \sim \mu\mathsf{X}.F(\mathsf{X})$. The guarded equation $\mathsf{X} = F(\mathsf{X})$ therefore has a solution, unique up to bisimilarity.*

Proof. Write $M = \mu\mathsf{X}.F(\mathsf{X})$.

Unfolding. The states of M are settled runtime terms of F and the states of $F(M)$ are runtime terms of F whose X-positions hold states of M, that is, settled runtime terms of F. Sending such a term to its settling is a coalgebra homomorphism from $F(M)$ to M: it preserves the halt map because settling does not change the outcome of a settled subterm and a term whose outcome is not running is already settled at the relevant positions, and it commutes with the step map because the step of a term is taken operator by operator and followed by settling, and settling is confluent (Theorem 4.21). Initial states correspond. A homomorphism is a bisimulation, so $F(M) \sim M$.

Uniqueness. Suppose $C \sim F(C)$ and $D \sim F(D)$; we show $C \sim D$, which gives the statement with $D = M$. Let

$$\mathcal{R} = \{(P, Q) : \text{there is a context } G \text{ with } P \sim G(C) \text{ and } Q \sim G(D)\}.$$

We show $\mathcal{R}$ is a bisimulation up to $\sim$, which suffices by the up-to technique for bisimilarity, valid here because $\sim$ is an equivalence and composing a bisimulation with $\sim$ on either side again yields a bisimulation.

Take $(P, Q) \in \mathcal{R}$ witnessed by G. If G contains no occurrence of X then $P \sim G \sim Q$ and there is nothing to prove. Otherwise substitute: $G(C) \sim G(F(C))$ by Theorem 5.1 applied inside G, and iterate the substitution. Because F is guarded, each substitution of F for X strictly increases the number of ticks that must be consumed before an X-position is reached; after finitely many substitutions the outermost X-positions of the resulting context G_n lie behind at least one tick. The halt maps of $G_n(C)$ and $G_n(D)$ then agree, because the outcome of a settled runtime term is determined by the constants that are reachable without consuming a tick, and those are the same on both sides. A transition of $G_n(C)$ on input i is matched by the corresponding transition of $G_n(D)$, and the residuals are of the form $G'(C)$ and $G'(D)$ for a common context G', hence $\mathcal{R}$-related up to $\sim$. The same argument runs in the other direction, so $\mathcal{R}$ is a bisimulation up to $\sim$ and $C \sim D$. □

Corollary 7.2 (loop unfolding)**.** *If $A\,;\mathsf{X}$ is guarded then $\mathsf{while}_b(A) \sim (A\,;\mathsf{while}_b(A)) \oplus_b \mathsf{skip}$.*

The guardedness hypothesis is not a technical convenience. It is the difference between an equation with a meaning and an equation without one, and the clearest instance is the retry loop.

Proposition 7.3 (retry needs a delay)**.** *Let $A : X \Rightarrow X$ have failure alphabet E and let A be able to fail at an initial state, that is, $\mathsf{init}_A(x)$ is failed for some x. Then the context $A \rhd \mathsf{X}$ is not guarded, and the equation $\mathsf{X} = A \rhd \mathsf{X}$ has no solution given by Theorem 4.20. For $d \geq 1$ the context $A \rhd (\mathsf{delay}_d\,;\mathsf{X})$ is guarded, so $\mu\mathsf{X}.(A \rhd (\mathsf{delay}_d\,;\mathsf{X}))$ exists and is unique up to bisimilarity.*

Proof. For the first claim, start the runtime term at such an x. The left component is failed, so the recovery rewriting applies and replaces it by the initial state of the right component, which is a pending X; unfolding produces a fresh copy of $A \rhd \mathsf{X}$ at the same value, whose left component is again failed. The $\rightsquigarrow$-chain is infinite and the context is not guarded. For the second, delay_d with $d \geq 1$ is engaged and does not halt before its first tick, so $\rho(A \rhd (\mathsf{delay}_d\,;\mathsf{X})) = 0$ by Theorem 4.23 and the existence and uniqueness follow from Theorem 7.1. □

Backoff is therefore not a tuning parameter of a retry loop. Without it the recursive equation that defines the loop has no solution in the model, and with a delay of at least one tick it has exactly one. Part IV builds its bounded retry combinator on this fact.

8 The guarded fragment

The guarded fragment of the signature consists of the constants skip and $\mathbf{0}$, the test agents $[b]$, sequential composition, guarded choice and the loop. Guarded Kleene Algebra with Tests [5] is the equational theory of exactly these constructs, with if and while primitive and the unrestricted sum and star absent. The system of that paper, Section 2, transcribed into the present notation and

numbered for reference here, consists of the guarded union axioms

$$e \oplus_b e \equiv e, \tag{1}$$
$$e \oplus_b f \equiv f \oplus_{\neg b} e, \tag{2}$$
$$(e \oplus_b f) \oplus_c g \equiv e \oplus_{b \wedge c} (f \oplus_c g), \tag{3}$$
$$e \oplus_b f \equiv ([b] \,;\, e) \oplus_b f, \tag{4}$$

the sequencing axioms

$$(e \,;\, f) \,;\, g \equiv e \,;\, (f \,;\, g), \tag{5}$$
$$\mathsf{skip} \,;\, e \equiv e \equiv e \,;\, \mathsf{skip}, \tag{6}$$
$$(e \oplus_b f) \,;\, g \equiv (e \,;\, g) \oplus_b (f \,;\, g), \tag{7}$$
$$\mathbf{0} \,;\, e \equiv \mathbf{0}, \qquad e \,;\, \mathbf{0} \equiv \mathbf{0}, \tag{8}$$

and the iteration axioms

$$\mathsf{while}_b(e) \equiv (e \,;\, \mathsf{while}_b(e)) \oplus_b \mathsf{skip}, \tag{9}$$
$$\mathsf{while}_c(e \oplus_b \mathsf{skip}) \equiv \mathsf{while}_c([b] \,;\, e), \tag{10}$$
$$g \equiv (e \,;\, g) \oplus_b f \;\text{ and } e \text{ productive } \implies g \equiv \mathsf{while}_b(e) \,;\, f. \tag{11}$$

Theorem 8.1 (the guarded axioms hold up to bisimilarity)**.** *Let T be any of* Id*, $\mathcal{P}$, $\mathcal{D}$. Interpret* skip*,* $\mathbf{0}$*,* $[b]$*,* $;$*,* $\oplus_b$ *and* while_b *by Theorems 4.1, 4.2, 4.6, 4.10, 4.11 and 4.22, and read productivity as guardedness. Then* (1) *to* (7) *and* (9) *to* (11) *hold with* $\equiv$ *read as* $\sim$*, on all terms whose loop bodies are guarded. Of the two annihilation axioms of* (8)*, the left one holds up to* $\sim$ *and the right one does not.*

Proof. (1). Both branches of $e \oplus_b e$ are copies of e, so the relation pairing each copy with the original is a bisimulation containing both possible initial pairs.

(2). The initial state of $e \oplus_b f$ at x is the initial state of e when $b(x) = 1$ and of f otherwise, and the same holds of $f \oplus_{\neg b} e$; the branch subcoalgebras are identical.

(3). At x the left side selects e when $c(x) = 1$ and $b(x) = 1$, f when $c(x) = 1$ and $b(x) = 0$, and g when $c(x) = 0$; the right side selects e when $(b \wedge c)(x) = 1$, and otherwise selects f when $c(x) = 1$ and g when $c(x) = 0$. The two case splits coincide.

(4). When $b(x) = 0$ both sides are f at x. When $b(x) = 1$ the left side is e at x, and $[b]$ at x is skip, which halts at its initial state with x; by the left unit law of Theorem 6.1, $[b] \,;\, e \sim e$ on that branch. Formally the graph of the injection of S_e into $S_{[b];e}$ restricted to $b(x) = 1$ is a bisimulation.

(5) and (6) are Theorem 6.1.

(7). At x with $b(x) = 1$, the left side starts e and, when e halts, hands to g; the right side is $e \,;\, g$ at x. The states of the two agents are in bijection compatible with the handoff, and the bijection is a coalgebra isomorphism. The case $b(x) = 0$ is symmetric.

Left annihilation in (8). $\mathbf{0}$ is running at its single state, so $\mathbf{0}^\circ = \{*\}$ and the initial state of $\mathbf{0} \,;\, e$ is $\mathsf{inl}(*)$, whose step is $\eta(\tau, \mathsf{inl}(*))$ and whose outcome is running. The singleton relation $\{(\mathsf{inl}(*), *)\}$ is a bisimulation onto $\mathbf{0}$.

(9) is Theorem 7.2.

(10). Both sides are defined only when their loop bodies are guarded. Suppose $e \oplus_b \mathsf{skip}$ is guarded as a loop body under the test c. If some value x reachable at the loop head had $c(x) = 1$ and $b(x) = 0$, the body would select skip, which halts at its initial state, the loop would hand back to its own head at the same value without consuming a tick, and the context would not be guarded.

Hence $b(x) = 1$ at every reachable loop-head value with $c(x) = 1$, and there $[b]\,;e \sim e$ as in (4). The two loops therefore have bisimilar bodies at every state they reach, and Theorem 5.1 applied to the recursion gives the equation.

(11). Suppose $g \sim (e\,;g) \oplus_b f$ with e guarded. By (9), (7) and (6),

$$\mathsf{while}_b(e)\,;f \sim ((e\,;\mathsf{while}_b(e)) \oplus_b \mathsf{skip})\,;f \sim (e\,;\mathsf{while}_b(e)\,;f) \oplus_b (\mathsf{skip}\,;f) \sim (e\,;(\mathsf{while}_b(e)\,;f)) \oplus_b f,$$

so $\mathsf{while}_b(e)\,;f$ solves the same guarded equation as g, and Theorem 7.1 identifies them.

Right annihilation in (8) fails; the witness is Theorem 8.2. □

Proposition 8.2 (the axioms with the diverging agent on the right)**.** *Let $a \in \Sigma$ with $a \neq \tau$ and let A be the agent that emits a on its first tick and then halts. Then $A\,;\mathbf{0} \not\sim \mathbf{0}$ and, for $T = \mathcal{P}$, $A \oplus \mathbf{0} \not\sim A$. Both identities hold under $\simeq_{\mathrm{acc}}$: $A\,;\mathbf{0} \simeq_{\mathrm{acc}} \mathbf{0}$ and $A \oplus \mathbf{0} \simeq_{\mathrm{acc}} A$, both for every agent A.*

Proof. $A\,;\mathbf{0}$ emits a on its first tick, while $\mathbf{0}$ emits τ; no relation containing the pair of initial states can satisfy the transfer condition, since the two step values carry different output symbols. For the choice, the merged initial state of $A \oplus \mathbf{0}$ has step value $\{(a, \mathsf{inm}(\cdot)), (\tau, \mathsf{inr}(*))\}$ while A has step value $\{(a, \cdot)\}$; the Egli-Milner lifting of any relation requires every element of the first set to have a partner in the second with the same output symbol, and the element carrying τ has none.

For the accepted-trace statements: $A\,;\mathbf{0}$ never halts, because $\mathbf{0}$ never does, so it has no accepted trace, and neither has $\mathbf{0}$. For the choice, the identity is the union law of Theorem 9.1, proved in the next section, together with $\mathsf{acc}(\mathbf{0}) = \emptyset$. □

Remark 8.3. The left and right annihilation axioms behave differently because divergence is observable on the left of a composition and silent on the right. $\mathbf{0}\,;e$ never reaches e, so it is $\mathbf{0}$ on the nose; $e\,;\mathbf{0}$ performs all of e and then falls silent, and everything e emitted is still in the trace. Part I fixes $\mathbf{0}$ as divergence rather than as an empty choice for this reason, and the consequence is that every law mentioning $\mathbf{0}$ on the right of a composition is an accepted-trace law.

Remark 8.4. The tests behave as a Boolean algebra without any appeal to the unguarded choice. For tests b, c, d on X one has $[b]\,;[c] \sim [b \wedge c]$, $[b]\,;[b] \sim [b]$, $[b]\,;[\neg b] \sim \mathbf{0}$, $[1] = \mathsf{skip}$, $[0] = \mathbf{0}$, $[b] \oplus_c [d] \sim [(b \wedge c) \vee (d \wedge \neg c)]$ and $[b] \oplus_b [\neg b] \sim \mathsf{skip}$. Each is immediate from Theorem 4.11: at a value where the guard holds the test agent is skip and the left unit law applies, and where it fails the test agent is $\mathbf{0}$ and the left annihilation law applies. The join of two tests is realised by a guarded choice, not by a sum, which is why the Boolean structure survives at bisimilarity while the sum does not.

9 Choice and accepted traces

Bisimilarity is too fine for the Kleene-algebra laws, and Part I supplies the coarser relation at which they hold. Three facts bound what can be proved there: left distributivity fails at bisimilarity, accepted-trace equality is not a congruence for sequential composition, and the algebra of accepted-trace sets has no multiplicative unit. Throughout, $T = \mathcal{P}$.

9.1 Accepted traces of the operators

Part I defines an accepted trace by its first halting tick: a run on an input word of length n is accepting with value y when the outcome after tick n is $\mathsf{inl}(y)$ and the outcome after every earlier tick is running, and $\mathsf{acc}(A, x)$ collects the triples (w, σ, y) so obtained. An agent that never halts therefore has an empty accepted-trace set. An accepted trace also has length at least one, because

outcomes are read after ticks and not before the first, so an agent already halted at its initial state is recorded as accepting at tick 1 having emitted τ, and $\mathsf{acc}(\mathsf{skip}_X, x) = \{(i, \tau, x) : i \in I\}$. Both properties carry the weight of the proofs below.

Lemma 9.1 (accepted traces of the operators)**.** *Let $T = \mathcal{P}$ and write $\mathsf{acc}(A) = \{(x, w, \sigma, y) : (w, \sigma, y) \in \mathsf{acc}(A, x)\}$.*

1. *$\mathsf{acc}(A \oplus B) = \mathsf{acc}(A) \cup \mathsf{acc}(B)$, for all A and B and with no further hypothesis.*
2. *$\mathsf{acc}(A \oplus_b B) = \{q \in \mathsf{acc}(A) : b(x_q) = 1\} \cup \{q \in \mathsf{acc}(B) : b(x_q) = 0\}$, where x_q is the start value of q.*
3. *If A and B are engaged then $A\,;B$ is engaged and $\mathsf{acc}(A\,;B) = \mathsf{acc}(A) \cdot \mathsf{acc}(B)$, where*

$$\mathsf{acc}(A) \cdot \mathsf{acc}(B) \;:=\; \{(x, w_1 w_2, \sigma_1 \sigma_2, z) : (x, w_1, \sigma_1, y) \in \mathsf{acc}(A),\ (y, w_2, \sigma_2, z) \in \mathsf{acc}(B)\}.$$

4. *$\mathsf{acc}(\mathbf{0}) = \mathsf{acc}(\mathsf{fail}_e) = \emptyset$, $\mathsf{acc}(\mathsf{skip}_X) = \{(x, i, \tau, x)\}$, $\mathsf{acc}([b]) = \{(x, i, \tau, x) : b(x) = 1\}$, and $\mathsf{acc}(\mathsf{delay}_d) = \{(x, w, \tau^d, x) : |w| = d\}$ for $d \geq 1$.*
5. *The engaged agents are closed under $;$, $\oplus$ and $\oplus_b$, and on them $\simeq_{\mathrm{acc}}$ is a congruence for all three.*

Proof. (1) The merged initial state is running, so no run is accepting at tick 0 on either side, and the question is only about ticks $n \geq 1$. The first step of the merged state is by definition the union of the first steps of A and B taken from their initial states, and after that step a run stays inside one branch, with the outcomes of that branch. A run of $A \oplus B$ accepting at n is therefore a run of A or of B accepting at n, and conversely. When a branch is not engaged, its initial state is halted or failed and stutters, so the branch contributes exactly the runs A itself contributes, which are accepting at tick 1 or not at all; the two sides again agree. No engagement hypothesis is needed, because Part I reads the first outcome after the first tick on both sides.

(2) The guarded choice selects a branch at the initial state and is that branch thereafter.

(3) Let A and B be engaged. The initial state of $A\,;B$ is $h_{A,B}(\mathsf{init}_A(x)) = \mathsf{inl}(\mathsf{init}_A(x))$, which is running, so $A\,;B$ is engaged. A run of $A\,;B$ accepting at n passes through a unique position j at which the handoff occurs, the first position at which the underlying run of A is halted; before j the composite is running because A is, and from j onward it is B started at the value A halted with. Since B is engaged, position j is running in the composite, so $j < n$ and the B-part has length $n - j \geq 1$. The emitted word is the concatenation because the handoff is silent. Conversely two accepting runs that meet at a value compose. Both directions use $j \geq 1$, which holds because A is engaged.

(4) $\mathbf{0}$ and fail_e never halt, so no run is accepting. skip_X is halted at its initial state, so its outcome after tick 1 is $\mathsf{inl}(x)$ and the run emits τ; after tick $n \geq 2$ the earlier outcome is not running, so no longer run is accepting. The test agent is skip where b holds and $\mathbf{0}$ elsewhere. delay_d is running after ticks 1 to $d - 1$ and halted after tick d.

(5) Closure is part of (3) for $;$, is immediate for $\oplus$ because the merged initial state is running, and holds for $\oplus_b$ because each branch is engaged. For the congruence, (1) and (2) determine the accepted traces of a choice from those of its arguments. For $;$, let $A \simeq_{\mathrm{acc}} A'$ with both engaged; by (3), $\mathsf{acc}(A\,;B) = \mathsf{acc}(A) \cdot \mathsf{acc}(B) = \mathsf{acc}(A') \cdot \mathsf{acc}(B) = \mathsf{acc}(A'\,;B)$ for engaged B, and symmetrically in the second argument. □

The engagement hypothesis in clause (3) is not cosmetic, and dropping it breaks more than the clause.

Counterexample 9.2 (accepted-trace equality is not a congruence for sequencing). Let $T = \mathrm{Id}$. Then $\mathsf{skip}_X \simeq_{\mathrm{acc}} \mathsf{delay}_1^X$, while $\mathsf{skip}_X \,;\, \mathsf{delay}_1^X \not\simeq_{\mathrm{acc}} \mathsf{delay}_1^X \,;\, \mathsf{delay}_1^X$. Accepted-trace equality is therefore not a congruence for sequential composition on all agents, and the quotient of all agents by $\simeq_{\mathrm{acc}}$ is not an algebra.

Proof. skip_X is halted at its initial state, so its outcome after tick 1 is $\mathsf{inl}(x)$ and $\mathsf{acc}(\mathsf{skip}_X, x) = \{(i, \tau, x) : i \in I\}$ by Theorem 9.1(4). delay_1^X is running at its initial state, emits τ on its first tick and is then halted with x, so its accepted-trace set is the same, and the two agents are accepted-trace equivalent. Substituting one for the other on the left of a composition separates them. $\mathsf{skip}_X \,;\, \mathsf{delay}_1^X \sim \mathsf{delay}_1^X$ by the left unit law of Theorem 6.1, so its accepted traces are the one-tick set above. In $\mathsf{delay}_1^X \,;\, \mathsf{delay}_1^X$ the first factor halts after tick 1, the handoff is silent, and the second factor is then running for one further tick before halting, so the composite is halted only after tick 2 and its accepted-trace set is $\{(w, \tau\tau, x) : |w| = 2\}$. The two sets are disjoint. □

The witness says something specific: accepted traces cannot see the difference between halting at once and halting after one silent tick, and sequential composition can, because the successor starts at the moment the predecessor halts. Restricting to engaged agents removes the blind spot, since an engaged agent halts after at least one tick and the accepted traces then record where.

Proposition 9.3 (the accepted-trace algebra has no unit)**.** *No agent A satisfies $\mathsf{acc}(A) = \{(x, \epsilon, \epsilon, x) : x \in X\}$, the unit for the concatenation of Theorem 9.1(3). Consequently the accepted-trace sets of engaged agents form a semiring without a multiplicative unit, the map* acc *does not send* skip *to that unit, and* acc *is not a homomorphism for* ; *on terms in which* skip *occurs.*

Proof. Every accepted trace of Part I has input word of length at least one, because the outcome that makes a run accepting is read after a tick. The empty element therefore lies in no accepted-trace set. For the consequence, $\mathsf{acc}(\mathsf{skip}_X)$ is the set of one-tick idle traces by Theorem 9.1(4), and for engaged A one has $\mathsf{acc}(\mathsf{skip} \,;\, A) = \mathsf{acc}(A)$ by Theorem 6.1, whereas $\mathsf{acc}(\mathsf{skip}) \cdot \mathsf{acc}(A)$ prefixes every element with a tick. □

The unit laws survive at the level of agents, where they are bisimilarities, and fail at the level of accepted-trace sets, where the unit does not exist. The same one tick reappears in every statement below that mentions skip inside a sum, including the iterate.

9.2 The idempotent semiring with tests

Theorem 9.4 (the star-free axioms hold under accepted traces)**.** *Let $T = \mathcal{P}$ and let Ag_e denote the engaged agents. Then Ag_e is closed under ;, $\oplus$ and $\oplus_b$, $\simeq_{\mathrm{acc}}$ is a congruence for those three operations on Ag_e, and $(\mathsf{Ag}_e/\simeq_{\mathrm{acc}},\ \oplus,\ \mathbf{0},\ ;)$ is an idempotent semiring without a multiplicative unit:*

$$
\begin{aligned}
&s \oplus (t \oplus u) \simeq_{\mathrm{acc}} (s \oplus t) \oplus u, \quad s \oplus t \simeq_{\mathrm{acc}} t \oplus s, \quad s \oplus s \simeq_{\mathrm{acc}} s, \quad s \oplus \mathbf{0} \simeq_{\mathrm{acc}} s,\\
&(s \,;\, t) \,;\, u \simeq_{\mathrm{acc}} s \,;\, (t \,;\, u), \quad \mathbf{0} \,;\, s \simeq_{\mathrm{acc}} \mathbf{0} \simeq_{\mathrm{acc}} s \,;\, \mathbf{0},\\
&s \,;\, (t \oplus u) \simeq_{\mathrm{acc}} (s \,;\, t) \oplus (s \,;\, u), \quad (s \oplus t) \,;\, u \simeq_{\mathrm{acc}} (s \,;\, u) \oplus (t \,;\, u).
\end{aligned}
$$

Sequential composition has skip *as a two-sided unit on all agents, up to $\sim$ and hence up to $\simeq_{\mathrm{acc}}$. The test agents satisfy $[b] \,;\, [c] \sim [b \wedge c]$, $[b] \,;\, [b] \sim [b]$, $[b] \,;\, [\neg b] \sim \mathbf{0}$, $[1] = \mathsf{skip}$, $[0] = \mathbf{0}$ and $[b] \oplus [\neg b] \simeq_{\mathrm{acc}} \mathsf{skip}$, so they form a Boolean algebra with meet given by composition and join by the sum. Every identity of the star-free axioms of Kleene algebra with tests is therefore valid for this interpretation, and equational reasoning from them is sound as long as substitution stays inside Ag_e, which Theorem 9.2 shows is necessary.*

Proof. Closure and congruence are Theorem 9.1(5). The four axioms for $\oplus$ are the corresponding properties of union, since $\mathsf{acc}(\oplus)$ is union by Theorem 9.1(1) and $\mathsf{acc}(\mathbf{0}) = \emptyset$ by (4). Associativity of ; and the two unit laws hold up to $\sim$ by Theorem 6.1 and descend to $\simeq_{\mathrm{acc}}$ because bisimilar agents have equal accepted traces. Annihilation on the left holds up to $\sim$ by Theorem 8.1; on the right, $s\,;\mathbf{0}$ never halts, so both sides have empty accepted-trace sets. The two distributive laws follow from Theorem 9.1(1) and (3): both sides have accepted-trace set $\mathsf{acc}(s)\mathsf{acc}(t) \cup \mathsf{acc}(s)\mathsf{acc}(u)$ and $\mathsf{acc}(s)\mathsf{acc}(u) \cup \mathsf{acc}(t)\mathsf{acc}(u)$ respectively, and concatenation of languages distributes over union on both sides. The identities among test agents up to $\sim$ are the remark closing Section 8. For the join, $\mathsf{acc}([b] \oplus [\neg b])$ is $\mathsf{acc}([b]) \cup \mathsf{acc}([\neg b])$ by clause (1), which is the set of one-tick idle traces at every start value, which is $\mathsf{acc}(\mathsf{skip})$ by clause (4). □

Iteration is available in the fragment, and it carries the tick that Theorem 9.3 predicts.

Proposition 9.5 (the iterate and its exit tick)**.** *Let A be engaged, so that $A\,;\mathsf{X}$ is guarded, and put $A^{\circledast} := \mu\mathsf{X}.((A\,;\mathsf{X}) \oplus \mathsf{skip})$. Then $A^{\circledast}$ is engaged, the unfolding law $A^{\circledast} \sim (A\,;A^{\circledast}) \oplus \mathsf{skip}$ holds, and*

$$\mathsf{acc}(A^{\circledast}) \;=\; \bigcup_{k\geq 0} \mathsf{acc}(A)^k \cdot \mathsf{acc}(\mathsf{skip}).$$

The induction axiom of Kleene algebra fails for this iterate: $\mathsf{acc}(A^{\circledast}\,;B)$ contains one idle tick more than $\bigcup_{k\geq 0} \mathsf{acc}(A)^k \cdot \mathsf{acc}(B)$ at every k.

Proof. A is engaged, so it halts after at least one tick, so no silent path through $A\,;\mathsf{X}$ reaches the hole and the context is guarded by Theorem 4.23; the unfolding law is Theorem 7.1. The initial state is the merged state of the choice, which is running, so the iterate is engaged. For the accepted traces, unfold k times and take the skip branch. Each round runs A to a halt and hands its value silently to the next loop head, which contributes $\mathsf{acc}(A)^k$ by the handoff analysis of Theorem 9.1(3). At the loop head the choice state is running, and the tick that selects the skip branch leaves the composite halted, so the exit contributes exactly the one-tick set $\mathsf{acc}(\mathsf{skip})$ of Theorem 9.1(4). Summing over k with Theorem 9.1(1) gives the displayed identity, and every accepting run exits after some finite number of rounds. The last claim is Theorem 9.3 applied to the exit factor: $\mathsf{acc}(\mathsf{skip})$ is one idle tick rather than the concatenation unit. □

9.3 What fails at bisimilarity

Counterexample 9.6 (left distributivity). Let $T = \mathcal{P}$ and let a, b, c be distinct non-idle output symbols. Let A, B, C be the agents that emit a, b, c respectively on their first tick and then halt. Then $A\,;(B \oplus C) \not\sim (A\,;B) \oplus (A\,;C)$, although the two are $\simeq_{\mathrm{acc}}$-equal.

Proof. After its first tick, $A\,;(B \oplus C)$ is in the merged initial state of $B \oplus C$, a single state whose step value is $\{(b,\cdot),(c,\cdot)\}$. After its first tick, $(A\,;B) \oplus (A\,;C)$ is in one of two states, one with step value $\{(b,\cdot)\}$ and one with step value $\{(c,\cdot)\}$; the branch was selected by the first transition. Suppose R were a bisimulation. Egli-Milner lifting applied to the first transition of the right-hand agent requires the merged state of $B \oplus C$ to be related to one of the two branch states, and then the transfer condition on the next transition fails, because the merged state has an outgoing b and an outgoing c while each branch state has only one of them. The $\simeq_{\mathrm{acc}}$-equality is the left distributive law of Theorem 9.4, whose hypotheses hold because A, B and C are engaged. □

Operationally the two terms differ in when the branch is chosen, and therefore in what the choosing agent can have seen. In $(A\,;B) \oplus (A\,;C)$ the choice is made on the first tick and can depend only on the first input; in $A\,;(B \oplus C)$ it is made on the second and can depend on the

second input as well. A router that reads its input at the moment it commits distinguishes them, which is the practical reading of the counterexample.

Proposition 9.7 (the surviving structure at bisimilarity)**.** *Let $T = \mathcal{P}$ and restrict to engaged agents. Modulo $\sim$: $\oplus$ is associative, commutative and idempotent; $;$ is associative with* skip *as a two-sided unit; $(A \oplus B)\,;C \sim (A\,;C) \oplus (B\,;C)$; and $\mathbf{0}\,;A \sim \mathbf{0}$. Left distributivity fails by Theorem 9.6 and $\mathbf{0}$ is not a unit for $\oplus$ by Theorem 8.2. The structure up to bisimilarity is therefore one-sided: an idempotent commutative semigroup under $\oplus$, a monoid under $;$, with distributivity on the right only and no additive unit.*

Proof. Associativity and commutativity of $\oplus$: the merged initial state of either bracketing has as its first step the union of the first steps of the three summands from their initial states, and thereafter the run is inside one summand; the relation pairing corresponding summand states, together with the pair of merged initial states, is a bisimulation in both cases. Idempotence: for engaged A, the merged initial state of $A \oplus A$ has first step the union of two copies of $\mathsf{step}_A(\mathsf{init}_A(x), i)$, which is that set again, and $\mathsf{init}_A(x)$ is running, so pairing the merged state with $\mathsf{init}_A(x)$ and each copy of a state of A with itself is a bisimulation. Monoid laws are Theorem 6.1. Right distributivity: the merged initial state of $(A \oplus B)\,;C$ has first step the union of the first steps of A and B followed by the handoff map into C, which is the first step of the merged initial state of $(A\,;C) \oplus (B\,;C)$ transported along the evident bijection of state spaces; the bijection is a coalgebra isomorphism on the reachable parts. Left annihilation is proved in Theorem 8.1. □

Remark 9.8. One-sided axiomatisations of Kleene algebra are studied in their own right; Kozen and Silva [10] prove completeness for the left-handed star axioms, dropping the right-handed induction rule. The one-sidedness here is on the distributive rather than on the star side, and we do not claim their system is sound for $\sim$: the induction rules are inclusion statements, and the only iteration in the present signature is the guarded loop, whose laws are those of Theorem 8.1, and the unguarded iterate of Theorem 9.5, whose induction axiom fails.

Proposition 9.9 (probabilistic choice)**.** *Let $T = \mathcal{D}$ and let p, q be rational with $pq \neq 1$. Then, up to $\sim$, $A \oplus_p B \sim B \oplus_{1-p} A$; $A \oplus_p A \sim A$ for engaged A; $(A \oplus_p B) \oplus_q C \sim A \oplus_{pq} (B \oplus_r C)$ with $r = q(1-p)/(1-pq)$, again rational; and $(A \oplus_p B)\,;C \sim (A\,;C) \oplus_p (B\,;C)$.*

Proof. Each is a computation on the first step of the merged initial state, matched by the same state bijections as in Theorem 9.7, with Larsen-Skou lifting in place of Egli-Milner. For the third, the left-hand first step is qp on the first step of A, $q(1-p)$ on that of B and $1-q$ on that of C, and the right-hand first step is pq on A and $1-pq$ distributed as r and $1-r$ over B and C; substituting r gives $(1-pq)r = q(1-p)$ and $(1-pq)(1-r) = 1-q$. □

10 The synchronous product

The synchronous product runs two agents in lockstep on a product interface, the operator that Esterel-style languages make primitive [4]. It is a symmetric monoidal product up to bisimilarity and it violates the exchange law of concurrent Kleene algebra, which is why Part III introduces a different parallel operator rather than reusing this one.

Theorem 10.1 (the product is symmetric monoidal)**.** *Let T be commutative. Then up to $\sim$, and modulo the evident isomorphisms of interfaces, value types and failure alphabets,*

$$(A \otimes B) \otimes C \sim A \otimes (B \otimes C), \qquad A \otimes B \sim B \otimes A, \qquad A \otimes \mathsf{skip}_1 \sim A.$$

The same three laws hold for race *in place of $\otimes$.*

Proof. All three are witnessed by the canonical bijections of state spaces, which are coalgebra isomorphisms.

Associativity. The bijection $(s_1, s_2, s_3) \mapsto (s_1, (s_2, s_3))$ commutes with the step maps because the double strength of a monad is associative, $\mathsf{dst}(\mathsf{dst}(u, v), w) = \mathsf{dst}(u, \mathsf{dst}(v, w))$ modulo the associativity of the product, and with the halt maps because each clause of Theorem 4.14 depends only on the multiset of the three outcomes together with the recorded reasons, which the bijection preserves.

Commutativity. The bijection $(s_1, s_2) \mapsto (s_2, s_1)$ commutes with the step maps precisely because T is commutative: for a commutative monad the two ways of forming dst agree, so $\mathsf{dst}(u, v) = T(\mathrm{swap})(\mathsf{dst}(v, u))$. It commutes with the halt maps because Theorem 4.14 records a simultaneous failure as a pair rather than by choosing a side, and likewise Theorem 4.16 records a simultaneous halt as a pair.

Unit. skip_1 has one state, which is halted, so $A \otimes \mathsf{skip}_1$ has state space $S_A \times 1$, the product interface is $(I \times 1, \Sigma \times 1)$, and its halt map reports $\mathsf{inl}(y, *)$ exactly when A reports $\mathsf{inl}(y)$. The projection is a coalgebra isomorphism over the interface isomorphism. □

Remark 10.2. Commutativity of T is used and cannot be dropped. Replace T by the writer monad on the free monoid on two letters, which is not commutative, and the two evaluation orders of a product of two agents produce different accumulated words, so the swap bijection is no longer a coalgebra map. That monad is outside the fixed list of Part I; the point of the substitution is only to show that the hypothesis in Theorem 10.1 is doing work.

Counterexample 10.3 (the exchange law fails in both directions). Let $T = \mathrm{Id}$. Let a halt after one tick emitting a, let b halt after two ticks emitting b b, and let c and d halt after one tick emitting c and d. Then

$$(a\,;c) \otimes (b\,;d) \quad \text{and} \quad (a \otimes b)\,;(c \otimes d)$$

are not related by $\sqsubseteq$ in either direction.

Proof. Both agents are deterministic, so each has exactly one run per input word and refinement reduces to inclusion of stutter closures of single traces. On the left, the first component emits a at tick 1, hands to c, emits c at tick 2 and then halts and stutters; the second component emits b at ticks 1 and 2, hands to d, and emits d at tick 3. The product halts when both have halted, at tick 3, with trace $(\mathrm{a}, \mathrm{b})\,(\mathrm{c}, \mathrm{b})\,(\tau, \mathrm{d})$. On the right, $a \otimes b$ halts when both factors have halted, that is at tick 2, with trace $(\mathrm{a}, \mathrm{b})\,(\tau, \mathrm{b})$, and then $c \otimes d$ emits (c, d) at tick 3. The two traces

$$(\mathrm{a}, \mathrm{b})\,(\mathrm{c}, \mathrm{b})\,(\tau, \mathrm{d}) \quad \text{and} \quad (\mathrm{a}, \mathrm{b})\,(\tau, \mathrm{b})\,(\mathrm{c}, \mathrm{d})$$

differ at ticks 2 and 3 and neither is obtained from the other by inserting or deleting ticks carrying the idle pair, since the differing positions carry non-idle symbols in at least one component and the input word is arbitrary. Neither refines the other. □

The failure is a statement about waiting. A synchronous product does not release its halting value until both factors have halted, so a fast factor idles, and the idling is visible in the other component's output. Concurrent Kleene algebra [2] asks for the exchange inclusion $(p \parallel q)\,;(r \parallel s) \sqsubseteq (p\,;r) \parallel (q\,;s)$, which holds when the parallel operator interleaves independently rather than in lockstep. Part III builds such an operator from channels and proves the inclusion there. The equal-duration case is the exception.

Proposition 10.4 (interchange under synchronous halting)**.** *Let $A : X_1 \Rightarrow Y_1$ and $B : X_2 \Rightarrow Y_2$ halt at the same position on every input word and every pair of start values, that is, for all x_1, x_2*

and all w the run of A from x_1 is halted at position n if and only if the run of B from x_2 is. Then, for all C and D of the matching types,

$$(A\,;C)\otimes(B\,;D)\sim(A\otimes B)\,;(C\otimes D).$$

Proof. Under the hypothesis the two handoffs happen at the same position n on both sides. Before position n the two agents are $A\otimes B$ on the nose; at position n the left side replaces each component by the initial state of its successor, and the right side replaces the pair by the initial state of $C\otimes D$, which is the same pair. After n both are $C\otimes D$. The bijection (component-wise phase) $\mapsto$ (joint phase) is therefore a coalgebra isomorphism on reachable states, and the halt maps agree because both report a halt exactly when C and D have both halted. □

Counterexample 10.5 (sequencing does not distribute over the product). Let $T=\mathrm{Id}$ and let A emit a and halt after one tick. Then $A\,;(B\otimes C)$ and $(A\,;B)\otimes(A\,;C)$ differ: the first runs A once and the second runs two copies of it, so on the first tick the first emits (a) on a single-component interface while the second emits the pair (a,a), and the two live over different interfaces. Even after identifying the interfaces along the diagonal, the first agent's copy of A has one state and the second's has two, and the two disagree as soon as A is nondeterministic: with $T=\mathcal{P}$ and A emitting a or a', the first agent emits the same symbol into both components and the second may emit different ones.

A prefix cannot be duplicated across a product. Fan-out is not written by repeating a term; it is written by running the shared prefix once and delivering its value to both consumers, which requires the wiring of Part III rather than an operator of this Part.

11 Failure and recovery

Failure in this Part is an outcome of an agent, not an event of a runtime. An agent fails when its state is failed, the reason is drawn from the fixed finite alphabet E, and the only operator that observes a failure is recovery. Part IV's crashes are a different thing: they are inflicted by a scheduler and are recorded as marker outputs.

Proposition 11.1 (the recovery laws)**.** *For $A : X \Rightarrow Y$ with failure alphabet E, $B : E \Rightarrow Y$ with failure alphabet E' and $C : E' \Rightarrow Y$:*

$$(A\rhd B)\rhd C\sim A\rhd(B\rhd C),\qquad \mathsf{skip}_X\rhd B\sim\mathsf{skip}_X,\qquad \mathsf{fail}_e\rhd B\sim\iota_e\,;B,$$

where ι_e is the constant agent of Theorem 4.1. The same argument gives $\mathbf{0}\rhd B\sim\mathbf{0}$.

Proof. Associativity. In $(A\rhd B)\rhd C$, the composite $A\rhd B$ fails only when B fails, with B's reason, and A's failure is consumed by the inner recovery; so a failure of A with reason e leads to B started at e, and a subsequent failure of B with reason e' leads to C started at e'. In $A\rhd(B\rhd C)$ a failure of A with e leads to $(B\rhd C)$ started at e, which is B started at e and hands to C on a failure with e'. The canonical bijection of the three-fold sum of state spaces commutes with the two recovery maps, exactly as the handoff maps commuted in Theorem 6.1.

Units. skip_X has no failed state, so $S^{\dagger}_{\mathsf{skip}}=X$ and $\mathsf{skip}\rhd B$ has state space $X+S_B$ with the S_B part unreachable; the graph of the injection is a bisimulation onto skip_X. fail_e is failed at its single state, so the recovery fires before the first tick and the initial state of $\mathsf{fail}_e\rhd B$ is $\mathsf{inr}(\mathsf{init}_B(e))$; the same holds of $\iota_e\,;B$ by Theorem 4.6. $\mathbf{0}$ has no failed state, and the argument for skip applies. □

Counterexample 11.2 (recovery does not distribute over sequencing). Let $T = \mathrm{Id}$, let $A = \mathsf{fail}_e$, let B emit b and halt, and let C emit c and halt. Then $(A\,;B) \rhd C \not\sim (A \rhd C)\,;(B \rhd C)$.

Proof. On the left, $A\,;B$ is failed at its initial state, because a failure of the first component of a sequential composition propagates and B is never started. The recovery therefore fires before the first tick and the term is C started at e, which emits c at tick 1 and halts with C's value. On the right, $A \rhd C$ is also C started at e, so it halts at tick 1; the sequential composition then starts B at that value, and B emits b at tick 2 before halting with B's value. The left term is halted after tick 1 and the right is still running, so the two are not bisimilar. □

The two terms mean different things and the counterexample says which. On the left, the handler replaces the whole sequence; on the right, it replaces only the failing stage and the sequence resumes. Both are useful, and a framework that writes one and reasons about the other has a bug.

Remark 11.3. Which recovery laws hold is decided by the relation and not by the operators. The literal duplicate $(A \rhd B) \oplus (A \rhd B)$ is bisimilar to $A \rhd B$ for engaged arguments, by Theorem 9.7, and its accepted traces agree because union is idempotent. Two different handlers behave differently: when A fails, $(A \rhd B) \oplus (A \rhd C)$ has two accepted traces and one collection of outcomes, so an equivalence that reads the emitted words separates it from either summand and an equivalence that reads only outcomes does not.

12 Worked examples

The five patterns that agent frameworks implement are terms of the signature. Writing them down fixes what each one means and makes the laws above applicable to them.

Example 12.1 (handoff)**.** A pipeline in which a planner hands a plan to an executor and the executor hands a report to a critic is $\mathrm{Plan}\,;\mathrm{Exec}\,;\mathrm{Critic}$, unambiguously by Theorem 6.1. The value passed at each handoff is the halting value of the previous stage, and the handoff costs no tick, so the pipeline's duration is the sum of the stage durations and nothing else. If a stage fails, the pipeline fails with that stage's reason and the later stages are not started.

Example 12.2 (routing)**.** A router that inspects the incoming request and dispatches to one of two specialists is $\mathrm{Spec}_1 \oplus_b \mathrm{Spec}_2$ with b the routing test. The choice is made from the start value before the first tick, so the term is exactly as fast as the branch it selects, and (7) lets a common suffix be factored out of the two branches: $(\mathrm{Spec}_1 \oplus_b \mathrm{Spec}_2)\,;\mathrm{Report} \sim (\mathrm{Spec}_1\,;\mathrm{Report}) \oplus_b (\mathrm{Spec}_2\,;\mathrm{Report})$. Factoring a common *prefix* out is a different matter: Theorem 9.6 shows it changes the moment of decision when the choice is unguarded.

Example 12.3 (retry with backoff)**.** Three attempts with a delay of d ticks between them is

$$\mathrm{retry}_{3,d}(A) \;=\; A \rhd (\mathsf{delay}_d\,;(A \rhd (\mathsf{delay}_d\,;A))),$$

and the unbounded version is $\mu\mathsf{X}.(A \rhd (\mathsf{delay}_d\,;\mathsf{X}))$, which exists for $d \geq 1$ and does not exist for $d = 0$ when A can fail without consuming a tick (Theorem 7.3). The handler receives the failure reason, so a policy that retries only on some reasons is $A \rhd ((\mathsf{delay}_d\,;\mathsf{X}) \oplus_{\mathrm{retryable}} \mathsf{fail}_{\mathrm{giveup}})$ with the test applied to the reason.

Example 12.4 (loop)**.** An agent that thinks and acts until a goal test succeeds is the loop $\mathsf{while}_{\neg\mathrm{done}}(\mathrm{Think}\,;\mathrm{Act})$. By Theorem 4.22 it abbreviates $\mu\mathsf{X}.(((\mathrm{Think}\,;\mathrm{Act})\,;\mathsf{X}) \oplus_{\neg\mathrm{done}} \mathsf{skip})$. It is guarded exactly when one pass of think and act consumes a tick, which any agent that reads an

input does. The loop is the unique solution of its equation by Theorem 7.1, so two implementations of the same loop, for instance a recursive one and one with an explicit loop counter in the state, are bisimilar as soon as both satisfy the equation.

Example 12.5 (timeout)**.** A bounded wait is

$$\mathrm{timeout}_t(A) \;=\; \mathsf{race}(A, \mathsf{delay}_t \,;\, \iota_{\mathrm{to}}) \,;\, (\mathsf{fail}_{\mathrm{timeout}} \oplus_b \mathsf{skip}),$$

where ι_{to} is the constant agent of Theorem 4.1 at a reserved value and b tests whether the race was won by that arm alone. The race halts as soon as one factor halts, so the term halts by tick t, and the guarded choice turns a win by the timer into a failure. Writing the right arm as $\mathsf{delay}_t \,;\, \mathsf{fail}_{\mathrm{timeout}}$ instead does not work: by Theorem 4.16 a race fails only when both factors have failed, so a failing timer arm leaves the race running for as long as A runs. Part IV takes the term above as its definition of a timeout rather than adding an operator.

Example 12.6 (fan-out)**.** Two consumers of one producer are not $\mathrm{Prod} \,;\, (\mathrm{Cons}_1 \otimes \mathrm{Cons}_2)$ with the producer duplicated, by Theorem 10.5. The producer must be run once and its value delivered twice, which is a wiring question: the term is $\mathrm{Prod} \,;\, \Delta \,;\, (\mathrm{Cons}_1 \otimes \mathrm{Cons}_2)$ where $\Delta : X \Rightarrow X \times X$ is the agent that halts at once with the pair (x, x). That agent is skip up to a relabelling of its halting value, so fan-out of a *value* is available in this Part; fan-out of a *stream* is not, and Part III supplies it with channels.

13 Limitations

The series claim is quoted again here, as every Part quotes it.

> Part I fixes the object: discrete time, an effect monad T drawn from the list Id, $\mathcal{P}$, $\mathcal{D}$, and agents as effectful Mealy coalgebras with typed termination and failure as a terminal outcome. Parts II to IV introduce the operators under additional assumptions stated in each Part and prove, for each operator, that it is well-defined on agents, which relation it respects, and which laws hold at that relation: Part II for ;, $\oplus$, $\otimes$, $\triangleright$ and μ; Part III for Tr, channels, $\|$ and Gate_H; Part IV for Sup, Persist and Replay; Part V introduces no operator and interprets policies, plans, options, shields and controllers as terms of Parts I to IV, proving properties of their closed loops. For each of sixteen fields the series gives an explicit translation of one named mechanism into a term of the algebra together with an adequacy result for that translation, which is a soundness theorem, a one-way simulation, or a proved limitation; no field is claimed to reduce to the algebra.

This Part proves the operator half of that claim for ;, the three choices, $\otimes$, race, $\triangleright$ and μ, under three added assumptions collected from the statements above: loop bodies are guarded, the arguments of $\oplus$ and $\oplus_p$ are engaged, and T is commutative.

The equational theory is sound and its completeness is open. Theorem 8.1 says the guarded axioms hold; it does not say that every bisimilarity between guarded terms is derivable from them. The decision procedure of Smolka et al. [5] decides the free theory, and whether it decides bisimilarity of agent terms over a fixed set of constants is not settled here.

The Kleene structure is weaker than a Kleene algebra with tests, and three proved results say how much weaker. The accepted-trace algebra has no multiplicative unit (Theorem 9.3). Accepted-trace equality is not a congruence for sequential composition (Theorem 9.2). Left distributivity fails at bisimilarity (Theorem 9.6). What survives is the star-free semiring with tests of Theorem 9.4 on the engaged fragment, together with the iterate of Theorem 9.5. That iterate satisfies the unfolding law and fails the induction axiom by one tick, so an equational treatment of iteration has to use the guarded loop and the guarded axioms instead.

Remark 13.1. Commutativity of the product is a property of the semantics, and the execution models built on top of it in later Parts need not inherit it. The synchronous product steps both factors at every tick, so a scheduler that realises it is symmetric as well. The asynchronous interleaving of Part III is not: a scheduler that always runs its left component first produces one word for A before B and another for B before A, and only a fair enumeration of schedules recovers the same word set. Nothing in Theorem 10.1 constrains a scheduler, which is the subject of Part IV.

Guardedness is decided by a sufficient syntactic criterion. Theorem 4.23 rejects some contexts that in fact settle, for instance one whose unguarded branch is unreachable at every value the loop can reach. A complete criterion would need reachability information, which is not part of the term.

Two whole subjects are deliberately absent. There is no operator here that lets two agents communicate while both are running; every operator either sequences, chooses, or runs in lockstep with no channel between the factors, and channels are Part III. And there is no notion of an agent being crashed and restarted by something outside it; the failures of this Part are the agent's own, and inflicted failure is Part IV.

The effect monad is one of three, all commutative. State and input-output effects are not commutative, so the product is not defined for them, and a premonoidal treatment is open. The results are stated per instance and no law here ranges over a combined nondeterministic and probabilistic model.

14 Conclusion

Agents and sequential composition form a category, bisimilarity is a congruence for every operator of the signature, and guarded equations have unique solutions. On the guarded fragment the algebra is a model of Guarded Kleene Algebra with Tests up to bisimilarity for each of the three effect monads, with the loop bodies required to consume a tick, and the tests form a Boolean algebra whose join is guarded choice. The synchronous product is symmetric monoidal and does not satisfy the exchange law.

The negative results are the ones that constrain design. Typed termination, which is what makes an agent different from a process, is also what prevents the unguarded choice from behaving like a sum: an agent that can stop now cannot also continue, so an accepted trace is never empty, the accepted-trace algebra has no multiplicative unit, and accepted-trace equality is not a congruence for sequencing. Guardedness, which is what makes recursion meaningful, is also what makes a retry loop well defined only when a delay separates the attempts. Neither observation is available at the level of the operators alone; both come from the object of Part I meeting the operators of this one.

Every later Part inherits the three hypotheses these theorems carry. A wiring operator, a runtime that crashes and restarts a term, and a policy written in this signature are all constructions over the operators defined here, so each of them is well defined up to bisimilarity only where loop bodies consume a tick, the arguments of an unguarded choice are running at their initial states, and the effect monad is commutative. Dropping any of the three does not weaken a later theorem; it removes the object the theorem is about.

References

[1] J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109-137, 1984.

[2] C. A. R. Hoare, B. Moller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. *Journal of Logic and Algebraic Programming*, 80(6):266-296, 2011. DOI: 10.1016/j.jlap.2011.04.005. The exchange law is the axiom relating the two compositions, Section 2.

[3] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[4] G. Berry and G. Gonthier. The Esterel synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87-152, 1992. DOI: 10.1016/0167-6423(92)90005-V.

[5] S. Smolka, N. Foster, J. Hsu, T. Kappe, D. Kozen, and A. Silva. Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time. *Proceedings of the ACM on Programming Languages*, 4(POPL), article 61, 2020. DOI: 10.1145/3371129. arXiv:1907.05920. The axiom system used in Section 8 is the one introduced in Section 2.

[6] J. Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67-111, 2000. DOI: 10.1016/S0167-6423(99)00023-4.

[7] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366-390, 1994. DOI: 10.1006/inco.1994.1037.

[8] D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427-443, 1997. DOI: 10.1145/256167.256195.

[9] D. Kozen. Typed Kleene algebra. Technical Report 98-1669, Computer Science Department, Cornell University, March 1998.

[10] D. Kozen and A. Silva. Left-handed completeness. In *Relational and Algebraic Methods in Computer Science (RAMiCS)*, LNCS 7560, pages 162-178. Springer, 2012. DOI: 10.1007/978-3-642-33314-9_11.

[11] A. McIver, T. Rabehaja, and G. Struth. Probabilistic concurrent Kleene algebra. In *Quantitative Aspects of Programming Languages (QAPL)*, EPTCS 117, pages 97-115, 2013. arXiv:1306.2697.

[12] J. Misra and W. R. Cook. Computation orchestration: a basis for wide-area computing. *Software and Systems Modeling*, 6(1):83-110, 2007.

[13] G. Plotkin and J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11:69-94, 2003. DOI: 10.1023/A:1023064908962.

[14] G. Plotkin and M. Pretnar. Handlers of algebraic effects. In *European Symposium on Programming (ESOP)*, LNCS 5502, pages 80-94. Springer, 2009. DOI: 10.1007/978-3-642-00590-9_7.

[15] D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011. The unique-solution argument of Theorem 7.1 uses the bisimulation up-to-bisimilarity technique presented there.

A Code evidence

The claims of this Part rest on their proofs. An executable model accompanies them, and this appendix records which of its checks bear on which claim. The model is finite: state spaces are explored to a stated cap and a stated tick budget, recursion is unfolded to a bounded depth, the distribution monad uses exact rational weights, and the three effect instances are handled separately and never combined. A finite-model check is a property tested over generated finite agents (100 runs, seed 20260903); an exhaustive bounded check enumerates all inputs or all resolutions up to a stated bound. The run these rows describe is recorded in `reviews/agent-algebra-vitest.txt`. Neither establishes a theorem, and the table says which is which. Claims not listed here have no

code row and rest on their proofs alone; that includes Theorem 9.2, Theorem 9.3, Theorem 9.5, Theorem 9.9, Theorem 10.5 in its nondeterministic form, and the general statement of Theorem 5.1 over all contexts.

Table 2: Finite-model and exhaustive bounded checks that accompany the results of this Part. Test titles are quoted from the model's law index; every test file path is relative to `code/agent-algebra`.

Claim (label)	File and identifier	Test (file › describe › it)	Runs	What the test shows
Theorem 6.1	`src/rules.ts` `stepSeq`; `src/ops.ts` `seq`, `skip`	`test/part2-composition-laws.test.ts` › sequential-category › seq is associative and skip is a two-sided unit up to bisimilarity	100, seed 20260903	Finite-model check: on generated triples of finite agents, associativity and both unit laws hold up to bisimilarity with the silent handoff in place.
Theorem 5.1, the ; and $\otimes$ cases	`src/rules.ts` `stepSeq`; `src/ops.ts` `seq`, `tensor`	`test/part2-composition-laws.test.ts` › sequential-category › bisimilar arguments give bisimilar composites under seq and tensor	100, seed 20260903	Finite-model check of two cases of the congruence theorem; the general statement over all contexts rests on its proof.
Theorem 8.1	`src/ops.ts` `choiceB`, `whileB`, `seq`, `zero`	`test/part2-composition-laws.test.ts` › gkat › every GKAT axiom holds up to bisimilarity on random deterministic agents	100, seed 20260903	Finite-model check, on generated deterministic agents, of the guarded union axioms (1) to (3), the sequencing axioms (5) to (7) and the loop unfolding (9). The logged property does not cover (4), (10) and (11), which rest on their proofs, and it covers the $T = \mathrm{Id}$ instance only.
Theorem 8.1, the guardedness hypothesis	`src/ops.ts` `whileB`, `fix`	`test/part2-composition-laws.test.ts` › gkat › the while fixed-point axiom fails on an unguarded body	All inputs to the stated bound	Exhaustive bounded check that the loop axiom is lost when the body halts without consuming a tick, so the hypothesis is not decorative.

Claim (label)	File and identifier	Test (file › describe › it)	Runs	What the test shows
Theorem 8.2	`src/ops.ts` `zero`, `seq`, `choiceND`; `src/equiv.ts` `acceptedTraceSet`	`test/part2-composition-laws.test.ts` › gkat › the axioms involving zero hold under accepted traces and fail up to bisimilarity	All inputs to the stated bound	Exhaustive bounded check of both directions on the fixed witnesses for $A\,;\mathbf{0}$, using guarded choice over generated emitters: accepted-trace equality holds and bisimilarity fails. The unguarded case $A \oplus \mathbf{0}$ is not in the logged test and rests on its proof.
Theorem 9.6	`src/ops.ts` `choiceND`, `seq`	`test/part2-composition-laws.test.ts` › kat-traces › left distributivity of seq over nondeterministic choice fails up to bisimilarity on the fixed witness	All resolutions to the stated bound	Exhaustive bounded check of the three-symbol witness: the two terms are not bisimilar.
Theorem 9.4	`src/ops.ts` `choiceND`, `seq`; `src/equiv.ts` `acceptedTraceSet`	`test/part2-composition-laws.test.ts` › kat-traces › left distributivity holds under finite-trace equivalence on random P agents	100, seed 20260903	Finite-model check of the distributive law at finite-trace equivalence on 100 internally nondeterministic P terms, each argument itself a choice of two emitters. Finite-trace equivalence is finer than the accepted-trace equality of the theorem, so the row bears on one axiom and the remaining axioms rest on their proofs.
Theorem 10.1	`src/ops.ts` `tensor`, `skip`; `src/effects.ts` `eprod`	`test/part2-composition-laws.test.ts` › monoidal › tensor is associative, commutative and unital up to bisimilarity	100, seed 20260903	Finite-model check of the three monoidal laws on generated agents for each of the three effect instances.

Claim (label)	File and identifier	Test (file › describe › it)	Runs	What the test shows
Theorem 10.1, the commutativity hypothesis	`src/effects.ts` `writerProdLeft`, `writerProdRight`; `src/ops.ts` `tensor`	`test/part2-composition-laws.test.ts` › monoidal › a non-commutative writer monad makes the two evaluation orders of tensor differ	All inputs to the stated bound	Exhaustive bounded check that the free-monoid writer effect, which is outside the fixed list of Part I, breaks commutativity, so the hypothesis is load-bearing.
Theorem 10.3	`src/ops.ts` `tensor`, `seq`; `src/equiv.ts` `traceEqFinite`, `refines`	`test/part2-composition-laws.test.ts` › exchange-fails › the one-tick versus two-tick exchange witness gives different traces	All inputs to the stated bound	Exhaustive bounded check that the two terms of the counterexample have different traces and that neither refines the other.
Theorem 10.4	`src/ops.ts` `tensor`, `seq`; `src/equiv.ts` `traceEqFinite`	`test/part2-composition-laws.test.ts` › exchange-fails › exchange holds when both left factors halt at the same tick	All inputs to the stated bound	Exhaustive bounded check of the equal-duration case on value-insensitive Id emitters. The proposition quantifies over all agents whose halting times agree on every input and over all three effect instances, and rests on its proof.
Theorem 7.1	`src/ops.ts` `fix`, `whileB`; `src/rules.ts` `stepFix`, `settleFix`	`test/part2-composition-laws.test.ts` › unique-solutions › k-fold unrolling of a guarded recursion is bisimilar to the loop-state implementation	100, seed 20260903	Finite-model check that two implementations satisfying the same guarded equation are bisimilar, which is the decidable shadow of uniqueness.
Theorem 7.3	`src/ops.ts` `retry`, `fix`; `src/rules.ts` `settleFix`	`test/part2-composition-laws.test.ts` › unique-solutions › unguarded retry of an immediately failing agent exceeds any step budget	All bounds to the stated cap	Exhaustive bounded check that the unguarded retry term settles at no bound, which is the failure of guardedness made observable.

Claim (label)	File and identifier	Test (file › describe › it)	Runs	What the test shows
Theorem 11.1	`src/ops.ts` `recover`, `race`, `failT`	`test/part2-composition-laws.test.ts` › recover-laws › recover is associative with skip and fail units, and race is associative and commutative up to bisimilarity	100, seed 20260903	Finite-model check of the handler laws, including that the handler is started at the failure reason, and of the two race laws.
Theorem 5.1, the ▷ and `race` cases	`src/ops.ts` `recover`, `race`, `failT`	`test/part2-composition-laws.test.ts` › recover-laws › bisimilar arguments give bisimilar composites under recover and race	100, seed 20260903	Finite-model check of two further cases of the congruence theorem.
Theorem 11.2	`src/ops.ts` `recover`, `seq`, `failT`	`test/part2-composition-laws.test.ts` › recover-laws › (A;B) recover C is not bisimilar to (A recover C);(B recover C)	All inputs to the stated bound	Exhaustive bounded check of the witness with a left factor that fails at its initial state, which is the witness the text uses: the right-hand term resumes the sequence after recovering and the left-hand term does not.
Theorem 9.6, the routing reading	`src/leaves.ts` `routerLeaf`, `pureLeaf`; `src/ops.ts` `seq`, `choiceND`, `choiceB`	`test/counterexamples.test.ts` › C2 left-distributivity-router › left distributivity fails because the router observes a later world	All inputs to the stated bound	Exhaustive bounded check that moving the choice one tick later changes what the deciding agent has read.
Theorem 10.5	`src/ops.ts` `seq`, `tensor`	`test/counterexamples.test.ts` › C3 seq-not-distributing-over-tensor › seq does not distribute over tensor	All inputs to the stated bound	Exhaustive bounded check that duplicating a prefix across a product changes the term, so fan-out needs wiring.

Claim (label)	File and identifier	Test (file › describe › it)	Runs	What the test shows
Theorem 11.3	`src/ops.ts` `recover`, `choiceND`, `failT`; `src/equiv.ts` `traceEqRuns`, `outcomeEq`, `acceptedTraceSet`	`test/counterexamples.test.ts` › C4 recover-choice-idempotence › recover-choice is not idempotent under trace equivalence but is under outcome equivalence	All inputs to the stated bound	Exhaustive bounded check that the literal duplicate is idempotent under both relations, while a choice between two different handlers has two accepted traces and one collection of outcomes.
Theorem 13.1	`src/sim.ts` `leftBiased`, `allFair`; `src/ops.ts` `tensor`, `asyncPar`; `src/equiv.ts` `relabel`	`test/counterexamples.test.ts` › C1 parallel-noncommutative-under-leftBiased › the parallel product is commutative up to bisimilarity but its leftBiased execution is not	One designed witness, tick budget 8	Exhaustive bounded check that the synchronous product is symmetric in execution as well as in semantics, while the asynchronous interleaving of Part III is not under a left-biased scheduler and is under a fair enumeration.