Computable Dynamics for Asynchronous MPST: Lyapunov Descent, Critical Capacity, and Decision Procedures

Sam Hart Berman

Abstract

This paper develops two computable analyses of quotient dynamics for asynchronous MPST: a quantitative Lyapunov-descent analysis and an algorithmic finite-reachability decision analysis. A weighted Lyapunov measure supplies the quantitative analysis, and finite-reachability decision schemas over regular fragments supply the algorithmic analysis. The weighted measure is $W = 2 \cdot \Sigma\text{depth} + \Sigma\text{buffer}$. Productive steps strictly decrease W, which yields explicit productive-step bounds and scheduler-lifted total-step bounds under stated assumptions. Finite-reachability schemas then yield decidability for asynchronous subtyping, regular type equivalence, crash-stop tolerance, and branching feasibility. All core claims are mechanized in Lean 4. This paper builds directly on *Coherence for Asynchronous Buffered MPST*, providing a quantitative and decision layer.

1 Introduction

1.1 Background

The preceding manuscript defines the operational coherence invariant $\mathsf{Coherent}(G, D)$ as the local invariant kernel of interest.

In the MPST literature, "coherence" typically refers to global-type coherence and projectability conditions, with later refinements of projection criteria (Honda et al., 2008; Honda et al., 2016; Castagna et al., 2012; Majumdar et al., 2021). Logical lines also frame coherence as n-ary duality or proof compatibility (Carbone et al., 2015; Carbone et al., 2016; Carbone et al., 2017), and data-certification extensions remain in the same global-projection discipline (Toninho and Yoshida, 2017).

In *Coherence for Asynchronous Buffered MPST*, $\mathsf{Coherent}(G, D)$ defines the inherited operational invariant over local environments and buffered traces.

1.2 Contribution

This paper studies what can be computed from that kernel statically and during runtime. Our contribution is twofold. We first give explicit quantitative bounds, then develop uniform decidability transfer through one finite-state schema.

The quantitative branch uses a classical Lyapunov template from stability theory (Lyapunov, 1892). The method chooses a nonnegative potential and proves strict decrease under productive transitions. In stochastic-process terms, this aligns with Foster-style drift arguments and modern Markov-stability treatments that lift local decrease to global bounds (Foster, 1953; Meyn and Tweedie, 2009). The weighted measure W in this paper is the protocol-level potential for that template.

Here the Lyapunov potential is an energy-like ranking function on states, not physical energy. The core proof pattern is nonnegative potential plus strict decrease on productive steps, which yields bounded productive progress.

The scheduler assumptions play the role of a drift condition. Under a declared fairness profile, productive decrease has an expected directional effect along traces, which is what justifies lifting local decrease to conservative total-step bounds.

The algorithmic branch follows a standard regularity-to-decidability strategy from finite-state analysis. Regularity assumptions are used to bound reachable structure, then one terminating exploration yields decision procedures by soundness and completeness transfer (Karp and Miller, 1969; Brand and Zafiropulo, 1983; Hopcroft and Ullman, 1979; Baier and Katoen, 2008). What is new here is a single reusable schema that instantiates to

asynchronous subtyping, regular equivalence, crash tolerance, and branching feasibility without changing the core proof pattern.

Capacity thresholds follow a standard phase-boundary intuition in buffered and queue-like systems, where behavior classes change when load or backlog crosses a boundary. The standard point is qualitative phase separation between safe and unstable regions. What is new here is a theorem-level threshold boundary B_c with an exact interface-level characterization under explicit MPST assumptions.

Operationally, B_c is a regime split. Below B_c one behavior class is admitted by the theorem profile. Above B_c a different class appears and may require stronger conditions. The boundary statement is exact under the paper's assumptions.

An information-theoretic view helps interpret the decision layer. Regularity hypotheses induce finite summaries that preserve exactly the predicates this paper decides, which is a sufficiency claim rather than heuristic compression (Cover and Thomas, 2006). In that sense the finite exploration graph is an analysis channel with no decision loss for the targeted properties.

The paper presents a single operational kernel with two branch-specific lifts. Equation 1.1 summarizes this split: quantitative bounds use weighted-measure descent laws, while algorithmic decidability uses regular finite reachability and finite exploration.

$$\begin{aligned} &\mathcal{K}_{\mathrm{q}} \Longrightarrow W\text{-descent} \Longrightarrow \text{bounds}, \\ &\mathcal{K}_{\mathrm{a}} \Longrightarrow \text{finite exploration} \Longrightarrow \text{deciders}. \end{aligned} \tag{1.1}$$

where $\mathcal{K}_{\mathrm{q}}$ is the operational kernel with weighted-measure laws, and $\mathcal{K}_{\mathrm{a}}$ is the regularity kernel.

Figure 1.1 gives the visual summary of this split.

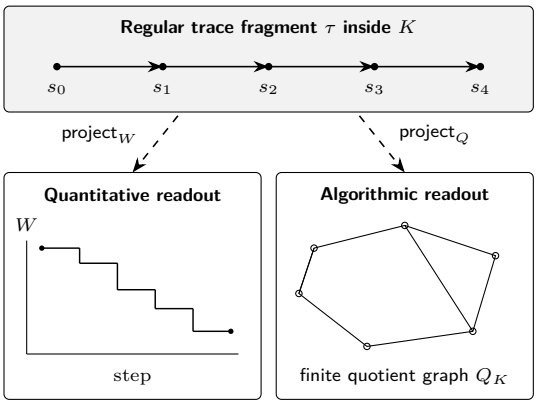

Figure 1.1: Two interpretations of one regular kernel fragment: quantitative descent and algorithmic quotient exploration.

1.3 Scope

Dependency across the three papers is explicit. *Coherence for Asynchronous Buffered MPST* supplies the invariant kernel, this paper supplies computable dynamics and decision procedures, and *Harmony from Coherence in Asynchronous MPST* lifts these components to reconfiguration and envelope theorems.

Scope is restricted to regular finite-reachability fragments for decision claims and explicit scheduler profiles for total-step bounds. Fault claims in this paper include crash-stop exactness and an explicit Byzantine safety characterization under a separate assumption bundle $\mathsf{H}_{\mathrm{byz}}$, in the broader distributed-fault context of FLP-style impossibility, partial synchrony, and failure-detector stratification (Fischer et al., 1985; Dwork, Lynch, and Stockmeyer, 1988; Chandra and Toueg, 1996), plus distributed-algorithm baselines (Lynch, 1996) and Byzantine baselines (Lamport et al., 1982; Castro and Liskov, 1999). Reconfiguration commutation and determinism-envelope maximality are deferred to *Harmony from Coherence in Asynchronous MPST*.

Result	Imported Dependencies and Use
Quantitative	Imports Paper 1 Theorem 4.1 and uses preserved Coherence for weighted descent and scheduler lifting.
Algorithmic	Imports the Paper 1 model kernel (Definitions 2.1–2.4) to fix state semantics for finite reachable constructions and deciders.
Crash	Imports the Paper 1 active-edge Coherence boundary to keep residual-graph checks in the same safety interface.
Byzantine	Imports the Paper 1 BZ interface set (BZ-1, BZ-1.1, BZ-1.2) and lifts these obligations to profile checks and capability-gated runtime admission.

Table 1.1: Series dependency map for Paper 2.

In Table 1.1, Quantitative, Algorithmic, Crash, and Byzantine correspond to Theorem 4.1, Theorem 5.3, Theorem 6.1, and Theorem 6.3 with Corollary 6.4 and Proposition 6.5. Byzantine progression note. Paper 1 provides the BZ interface obligations and their bridge-facing shape. This paper adds exact safety characterization at the declared profile interface, converse witness families, and profile-gated VM admission linkage.

Our contributions in this paper are as follows:

1. A quantitative descent theorem with explicit productive-step and scheduler-lifted total-step bounds from one weighted measure.
2. A uniform algorithmic schema that turns regular finite reachability into reusable decidability procedures.
3. A crash-stop tolerance characterization with decision and residual-graph exactness under stated side conditions.
4. A Byzantine safety characterization interface with exact iff form, converse counterexample families, and VM-bridge handoff obligations.
5. A mechanized architecture that shares one operational kernel while making branch-specific premises explicit (weighted-descent premises for quantitative claims, regular finite reachability premises for decision claims).

Main text proofs are proof sketches; full machine-checked derivations are provided by the Lean anchors indexed in Appendix F. Proof-sketch template used throughout the series: state proof mode (compositional/induction/coinduction/witness construction), give one concrete intermediate transformation, then conclude with the claim interface.

Related liveness lines often emphasize qualitative progress in session-typed settings (Honda et al., 2008, and Caires and Pfenning, 2010). Related mechanized lines expose proof brittleness and motivate reusable proof architecture (Tirore et al., 2025).

Symbol	Meaning
$\mathsf{H}_{\mathrm{byz}}$	Byzantine characterization bundle
$\mathsf{Obs}^{\mathrm{byz}}_{\mathrm{safe}}$	Byzantine safety-visible observation
$\mathsf{Eq}^{\mathrm{byz}}_{\mathrm{safe}}$	Byzantine safety-visible equality
$\mathcal{E}_{\mathrm{byz}}$	Byzantine determinism-envelope
Coherent	inherited operational coherence predicate
K	regular kernel fragment (shared source object in Fig. 1.1)
$\mathsf{project}_W$	projection from kernel fragment to weighted trajectory/readout
$\mathsf{project}_Q$	projection from kernel fragment to finite quotient exploration/readout
Q_K	finite quotient graph induced by kernel fragment K
W	weighted measure on state
W_0	initial weighted measure
ΔW	one-step change in weighted measure
$s,\ s_i$	configuration state(s) used in traces and bound statements
$P(\tau)$	productive-step count on trace τ
$T(\tau)$	total-step count on trace τ
$\mathsf{Profile}(\sigma)$	scheduler profile predicate
κ_σ	scheduler-lift factor for profile σ
$\mathsf{Phase}(B)$	low/critical/high phase at buffer level B
B_c	critical capacity threshold
N_s	explored state count in finite checker objects
N_e	explored transition-edge count in finite checker objects
$L_{\max}$	maximum active-edge buffered trace length
$W_{\max}$	maximum size of checked witness/certificate object
ByzChar	Byzantine characterization formula
ByzSafe	Byzantine safety predicate
DeliveryModel	parametric delivery interface

Table 1.2: Notation used in this paper.

2 Model Summary

The model uses asynchronous buffered semantics with active-edge Coherence from *Coherence for Asynchronous Buffered MPST*. Delivery behavior is parameterized by DeliveryModel.

Table 2.1 records the assumptions used for exact statements in this paper.

Assumption	Status
async buffered semantics	required
active-edge Coherence	required
regular finite-reachability	req. for decidability
fairness profile	req. for scheduler bounds
crash-stop fault model	req. for Theorem 6.1
Byzantine bundle $\mathsf{H}_{\mathrm{byz}}$	req. for Thm. 6.3, Cor. 6.4

Table 2.1: Assumptions for exact statements.

Assumption Block 2.1. Core Model Premises.
Core claims in this paper assume asynchronous buffered semantics, active-edge Coherence, regular finite reachability for decision results, and the stated fairness profile for scheduler-lifted bounds.

Assumption-block inheritance policy. Assumption blocks do not inherit implicitly. If a theorem does not name a block, it uses Assumption Block 2.1 only.

Definition 2.1. Well-Typed State and Step.
Write $\Gamma \vdash s\ \mathsf{wf}$ for state typing and $\Gamma \vdash s \to s'$ for one-step typing in the asynchronous buffered kernel reused from Paper 1.

Role split. Well-typedness gives rule admissibility and side-condition soundness for each step. Coherence gives the edge-local compatibility invariant transported across those typed steps.

Definition 2.2. Productive Step.
For the one-step transition relation $s \to s'$, a step is productive iff $W(s') < W(s)$, where W is Definition 2.3.

Relation to typing. Productive steps are a strict-descent subset of well-typed steps under the theorem premises. A well-typed step may be non-productive when it discharges administrative/profile obligations without decreasing W ($\Delta W = 0$).

Definition 2.3. Weighted Measure.
For a session state s,

$W(s) = 2 \cdot \Sigma\mathsf{depth}(s) + \Sigma\mathsf{buffer}(s)$. In the mechanization this function is `weightedMeasure`.

Definition 2.4. Depth Component.
Let depthLoc be the constructor count on one-step unfolded local types:

$$\mathsf{depthLoc}(\mathsf{end}) = 0.$$

$$\mathsf{depthLoc}(\mathsf{send}\ r\ T\ L) = 1 + \mathsf{depthLoc}(L),$$

$$\mathsf{depthLoc}(\mathsf{recv}\ r\ T\ L) = 1 + \mathsf{depthLoc}(L),$$

$$\mathsf{depthLoc}(\mathsf{select}\ r\ \{l_i : L_i\}_i) = 1 + \max_i d_i,$$

$$\mathsf{depthLoc}(\mathsf{branch}\ r\ \{l_i : L_i\}_i) = 1 + \max_i d_i,$$

where $d_i := \mathsf{depthLoc}(L_i)$.

$$\mathsf{depthLoc}(\mu L) = \mathsf{depthLoc}(\mathsf{unfold}(\mu L)).$$

For state s, $\Sigma\mathsf{depth}(s)$ is the sum of depthLoc over active endpoints. Guarded recursion premises ensure this quantity is finite on reachable states.

Coefficient choice. For $W_c(s) := c \cdot \Sigma\mathsf{depth}(s) + \Sigma\mathsf{buffer}(s)$, one productive send or select step satisfies $\Delta W_c \leq -c + 1$. Strict decrease requires $c > 1$. This paper fixes integer coefficient $c = 2$, which is the smallest integer that satisfies this requirement. The same coefficient is used for global lifts and scheduler bounds.

Table 2.2 lists scheduler profiles used by the bound statements.

Profile	Assumption	Bound class
round-robin	bounded delay	total conservative
k-fair	enabled within k slots	total conservative
adversarial fair	fairness only	productive exact

Table 2.2: Scheduler profiles.

Lift-constant extraction rules.

1. Round-robin profile: $\kappa_\sigma = N_{\max}$, where $N_{\max}$ is the profile bound on enabled roles between productive steps.
2. k-fair profile: $\kappa_\sigma = k$.
3. Adversarial-fair profile: fairness guarantees eventual scheduling of enabled productive steps, but does not bound non-productive interleavings. This paper therefore uses productive exactness only and does not assume a finite total-step lift constant for this row.

Interpretation note. "Productive exact" does not mean $\kappa_\sigma = 1$; it means the productive-step bound is exact while no finite profile-uniform total-step lift constant is claimed for the adversarial-fair row.

3 Worked Example

We trace the quantitative dynamics through an explicit witness. The example demonstrates strict Lyapunov descent across protocol steps, derives termination bounds via the scheduler-lift theorem, and illustrates phase classification at the critical capacity threshold. All numeric facts are computed from the quantitative rules in Section 4.

Running example. Protocol (as in *Coherence for Asynchronous Buffered MPST*):

```
C → P : Request(n)
P → C : Grant(k)
C → M : Report(k)
M → P : Confirm
P → C : Token(t)
```

Let the initial state be s_0 with one pending request on edge `(C,P)` and local depths 5, 5, and 2 for `P`, `C`, and `M`. Then

$$W_0 = W(s_0) = 2 \cdot (5 + 5 + 2) + 1 = 25. \quad (3.1)$$

Assume the typing/well-formedness judgment $\Gamma \vdash s_0\ \mathsf{wf}$ and step judgments $s_i \to s_{i+1}$ for the trace below.

Define witness trace

$$\tau_{\text{ex}} : s_0 \to s_1 \to s_2 \to s_3 \to s_4. \quad (3.2)$$

with step kinds `recv Request`, `send Grant`, `recv Grant`, `send Report`.

The quantitative facts are used through the following derivation forms.

Example derivation: descent.

$$\frac{\Gamma \vdash \tau_{\text{ex}} : s_0 \to^* s_4;\ \forall i < 4,\ \Delta W(s_i, s_{i+1}) = -1}{W(s_4) = W_0 - 4 = 21;\ P(\tau_{\text{ex}}) = 4 \leq W_0} \quad (3.3)$$

Example derivation: scheduler lift.

$$\frac{\mathsf{Profile}(\sigma);\ \kappa_\sigma = 2;\ W_0 = 25}{\forall \tau,\ T(\tau) \le \kappa_\sigma W_0 = 50} \tag{3.4}$$

For this witness, $T(\tau_{\mathrm{ex}}) = 4$, so the scheduler-lift upper bound gap is $50 - 4 = 46$. This illustrates the conservative profile-dependent margin.
Example derivation: phase boundary, using the phase function from Theorem C.3.

$$\frac{B_c = 2; \quad \mathsf{Phase}}{\begin{aligned}&\mathsf{Phase}(1) = \mathsf{Low}\\ &\mathsf{Phase}(2) = \mathsf{Critical}\\ &\mathsf{Phase}(3) = \mathsf{High}\end{aligned}} \tag{3.5}$$

Table 3.1 records the per-rule ΔW facts used in this witness.

Rule instance	ΔW
send `Report`	$-2+1=-1$
receive `Request`	-1
send `Grant`	$-2+1=-1$
receive `Grant`	-1

Table 3.1: Per-rule ΔW facts.

4 Quantitative Dynamics

Running-example thread. Section 3 instantiates this section with witness trace τ_{ex} and explicit per-step ΔW facts.

Assumption Block 4.1. Quantitative Descent Premises.
The descent theorem assumes well-typed asynchronous buffered transitions, active-edge Coherence preservation from *Coherence for Asynchronous Buffered MPST*, non-negativity of W, and a scheduler profile from Table 2.2 when total-step lifting is claimed.

Theorem 4.1. Quantitative Dynamics.
For any initial state s_0 and finite trace $\tau : s_0 \to \cdots \to s_n$ satisfying Assumption Block 4.1, let $P(\tau)$ be the productive-step count and $T(\tau)$ be the total-step count. Then

$$P(\tau) \le W(s_0) = W_0. \tag{4.1}$$

If the selected scheduler profile (fairness-required) provides lift constant κ_σ, then

$$T(\tau) \le \kappa_\sigma \cdot W_0. \tag{4.2}$$

The productive bound is exact as a tight upper bound. Equality is attained under the conditions stated below. The scheduler-lifted bound is conservative and profile-dependent.

Fairness-required clause. The productive bound is fairness-independent. The scheduler-lifted total-step clause requires the declared fairness profile used to extract κ_σ.

Premise roles. In Theorem 4.1, well-typedness determines the rule-local ΔW lemma applied at each step, and Coherence provides the invariant interface inherited from Paper 1.

Running-example instantiation. In Section 3, the witness trace τ_{ex} realizes this theorem with $W_0 = 25$, productive count $P(\tau_{\mathrm{ex}}) = 4$, and scheduler-lift gap reporting.

Proof. Write the trace as

$$s_0 \to s_1 \to \cdots \to s_n.$$

Let

$$I_{\mathrm{prod}} := \{\, i \in \{0, \ldots, n-1\} \mid W(s_{i+1}) < W(s_i) \,\}.$$

By definition, $P(\tau) = |I_{\mathrm{prod}}|$.

From the rule-local decrease lemmas (`send_step_decreases`, `recv_step_decreases`, `select_step_decreases`, `branch_step_decreases`) and their configuration lift `total_measure_decreasing`, each productive index $i \in I_{\mathrm{prod}}$ satisfies

$$W(s_i) - W(s_{i+1}) \ge 1.$$

Summing over productive indices gives

$$\begin{aligned}|I_{\mathrm{prod}}| &\le \sum_{i \in I_{\mathrm{prod}}} \big(W(s_i) - W(s_{i+1})\big)\\ &\le W(s_0) - W(s_n).\end{aligned}$$

Because Assumption Block 4.1 includes non-negativity of W, $W(s_n) \ge 0$, hence

$$P(\tau) = |I_{\mathrm{prod}}| \le W(s_0) = W_0.$$

This proves the productive-step bound.

For total-step lifting, fix a scheduler profile σ with certified lift constant κ_σ (Table 2.2). By the scheduler-lift lemma for that profile, total step

count is bounded by a profile factor times productive count:

$$T(\tau) \le \kappa_\sigma \, P(\tau).$$

Combining with $P(\tau) \le W_0$ yields

$$T(\tau) \le \kappa_\sigma \, W_0.$$

For the running witness from Section 3, this gives $T(\tau_{\mathrm{ex}}) = 4 \le 2 \cdot 25 = 50$, matching the conservative-gap calculation. Therefore both claimed inequalities hold. □

Proposition 4.2. Productive-Bound Equality Conditions.
Under Assumption Block 4.1, if a trace τ satisfies all of the following, then $P(\tau) = W_0$.

1. *Every productive step has unit drop, $W(s_i) - W(s_{i+1}) = 1$.*
2. *The terminal state has $W(s_n) = 0$.*

These conditions give the witness shape for bound attainment.

Proof sketch. The theorem proof gives

$$P(\tau) \le W(s_0) - W(s_n).$$

Under unit-drop and terminal-zero conditions, the telescoping sum over productive steps equals W_0. Hence $P(\tau) = W_0$. The key transformation is algebraic: rewrite the global bound as a telescoping sum over productive indices, then use the unit-drop hypothesis pointwise. This is a finite-sum argument (not a coinductive one). □

The weighted-measure decomposition used by Theorem 4.1 is: The next displays first separate state components in W, then restate the rule-family descent pattern that drives the global bound.

$$\begin{aligned} W(s) &= 2 \cdot \Sigma\mathrm{depth}(s) + \Sigma\mathrm{buffer}(s) \\ &\Downarrow \\ \mathsf{DepthTot}(s) &:= \Sigma\mathrm{depth}(s); \\ \mathsf{BufferTot}(s) &:= \Sigma\mathrm{buffer}(s); \\ W(s) &= 2\,\mathsf{DepthTot}(s) + \mathsf{BufferTot}(s). \end{aligned} \tag{4.3}$$

The rule-local descent pattern lifted to global productive-step bounds is:

productive rule family	net effect on W
send / select	$\Delta W \le -1$
recv / branch	$\Delta W \le -1$

(4.4)

$$\text{therefore} \quad \Delta W \le -1 \text{ on productive steps.} \tag{4.5}$$

5 Algorithmic Dynamics Schema

Running-example thread. Section 3 provides a finite witness trace inside the regular fragment used by this schema.

Assumption Block 5.1. Regular Finite-Reachability Premises.
The decision schema assumes regularity hypotheses that produce finite reachable-state spaces and effective predicate reductions to terminating finite-state checks on those spaces.

Definition 5.1. Regular Fragment Criteria.
A state belongs to the regular fragment when all of the following hold.

1. The role graph and message alphabet are finite.
2. Local types use guarded recursion and finite branch-label sets.
3. The closure of one-step unfolding and transition successors is finite.

Expressiveness note. This fragment includes finite-control request response workflows, guarded loop protocols, and finite branch families. It excludes unguarded recursion and constructions that generate unbounded fresh control states.

Definition 5.2. Admissible Predicate Families.
Write $\mathsf{AdmissiblePred}(\mathcal{P})$ when predicate family $\mathcal{P}$ admits a regular-fragment decision interface with all of the following.

1. Closure on explored objects: truth of $\mathcal{P}$ is invariant under the finite reachable-object normalization used by the checker.
2. Effective reduction interface: there are computable maps $\mathsf{obj}_{\mathcal{P}}$ and $\mathsf{check}_{\mathcal{P}}$ such that $\mathsf{obj}_{\mathcal{P}}(x)$ is finite under Assumption Block 5.1 and $\mathsf{check}_{\mathcal{P}}(x)$ terminates.
3. Reduction correctness: $\mathsf{check}_{\mathcal{P}}(x) = \mathsf{true} \iff \mathcal{P}(x)$.

Theorem 5.3. Algorithmic Dynamics Schema.
For any predicate family $\mathcal{P}$ such that Assumption Block 5.1 holds and $\mathsf{AdmissiblePred}(\mathcal{P})$*, there exists a total decider* $\mathsf{dec}_{\mathcal{P}}$ *such that*

$$\forall x,\ \mathsf{dec}_{\mathcal{P}}(x) = \mathsf{true} \iff \mathcal{P}(x). \tag{5.1}$$

In particular, asynchronous subtyping and regular type equivalence are decidable on the regular fragment.

Schema/instance boundary. Theorem 5.3 is a schema claim for families satisfying Definition 5.2. The instantiated claims in this paper are asynchronous subtyping and regular type equivalence only; other families require an explicit admissibility witness.

Running-example instantiation. The request/-grant/report witness in Section 3 is an instance where the finite-state checker side of this theorem is directly realizable.

Coinductive boundary note. Theorem 5.3 is self-contained for the regular finite-reachability fragment in this paper and does not require Paper 3 for correctness. Effect-level transport is restricted to the witness-carrying rational subclass via `rational_effect_transport_bridge`; witness-check soundness/completeness is provided by `check_rational_effect_witness_sound` and `check_rational_effect_witness_complete`; strict outside non-transportability is witnessed by `strict_boundary_witness_effect`. Concretely, the boundary excludes unbounded non-rational coinductive behaviors without finite witnesses. Paper 3 consumes this as a boundary interface and does not back-propagate new premises into this theorem.

Proof sketch. Proof mode: compositional reduction with finite exploration. By Definition 5.2, fix $\mathsf{obj}_{\mathcal{P}}$ and $\mathsf{check}_{\mathcal{P}}$ with closure/correctness witnesses. Build finite reachable triple/pair structures from regularity (`reachable_triples_finite`, `reachable_pairs_finite`), run terminating exploration/checkers, and apply reduction correctness. Instantiating this schema yields deciders for async subtyping (`async_subtype_decidable`) and regular equivalence (`regular_type_eq_check_sound`, `regular_type_eq_decide_spec`). Compositional part: finite-reachability construction plus terminating checker execution. Coinductive part: regular-equivalence correctness identifies a greatest fixed-point relation on the finite reachable graph, then proves the checker coincides with that relation by soundness/-completeness. □

Equation 5.2 records the generic workflow.

$$\begin{array}{c} \text{regularity assumptions} \\ \Downarrow \\ \text{finite reachable structure} \\ \Downarrow \\ \text{terminating checker } \mathsf{dec}_{\mathcal{P}} \\ \Downarrow \\ \mathsf{dec}_{\mathcal{P}}(x) = \mathsf{true} \iff \mathcal{P}(x). \end{array} \tag{5.2}$$

6 Decidability Layer

Running-example thread. Section 3 anchors the same protocol family used for crash and Byzantine interface readings in this section.

Assumption Block 6.1. Crash-Stop Characterization Premises.
Crash-tolerance characterization assumes crash-stop faults, finite role graphs, and the residual-graph side conditions used by the decision profile.

Theorem 6.1. Crash-Stop Tolerance Characterization.
Under Assumption Block 6.1, there exists a total decider $\mathsf{decCrash}$ *such that*

$$\forall (R, F),\quad \mathsf{decCrash}(R, F) = \mathsf{true} \iff \mathsf{CrashTolerant}(R, F). \tag{6.1}$$

Moreover crash tolerance satisfies an exact residual-graph characterization:

$$\mathsf{CrashTolerant}(R, F) \iff \mathsf{ResidualReachable}(R, F). \tag{6.2}$$

The theorem also includes witness families where topology changes flip tolerance outcomes within the stated side-condition class.

Running-example instantiation. For the running protocol shape in Section 3, crash removal is read as residual reachability over the same finite role graph.

Proof sketch. Construct the residual graph after removing crashed roles and edges, and define `crashTolerantDec` as reachability over that residual object. Soundness is provided by `crash_`

`tolerant_dec_sound`, which turns a positive decider result into crash tolerance. Completeness and exact characterization come from `crash_tolerance_iff`, and witness families are obtained by topology perturbations that cross the connectivity boundary. □

Assumption Block 6.2. Byzantine Characterization Premises.
Byzantine characterization assumes explicit bundle $\mathsf{H}_{\mathrm{byz}}$with fault-model, authentication and evidence-validity, conflict-exclusion primitive consistency, and adversarial-budget side conditions.

Shared Byzantine notation block (series baseline). This paper reuses ByzChar, ByzSafe, $\mathsf{Eq}^{\mathrm{byz}}_{\mathrm{safe}}$, and $\mathcal{E}_{\mathrm{byz}}$as defined at the Paper 1 interface, and only adds profile-indexed exactness packaging for the dynamics layer.

Definition 6.2. Byzantine Bundle Component Meanings.
At this paper interface:

1. ByzQuorumOK: quorum and intersection constraints hold for the declared fault budget. The standard threshold instantiation is $n > 3f$ with quorum intersection witnesses.
2. ByzAuthEvidenceOK: every accepted Byzantine-relevant action carries valid authentication and evidence objects required by the profile.
3. ByzBudgetOK: adversarial actions stay within the declared budget envelope.
4. ByzPrimitiveConsistent: conflict-exclusion and primitive-level consistency obligations hold for certified transitions.

ByzSafe means safety-visible non-conflict at the paper interface. No two admitted executions produce contradictory certified outcomes in $\mathsf{Obs}^{\mathrm{byz}}_{\mathrm{safe}}$for the same certified decision point.

Define

$$\begin{aligned}\mathsf{ByzChar} := \mathsf{ByzQuorumOK}\\ \wedge \mathsf{ByzAuthEvidenceOK}\\ \wedge \mathsf{ByzBudgetOK}\\ \wedge \mathsf{ByzPrimitiveConsistent}.\end{aligned} \tag{6.3}$$

where each conjunct names the corresponding requirement class in $\mathsf{H}_{\mathrm{byz}}$.

New in this paper: exact profile-indexed Byzantine safety characterization for the dynamics layer, plus converse witness families for dropped assumption classes.

Theorem 6.3. Exact Byzantine Safety Characterization.
Under Assumption Block 6.2, profile extraction yields an exact-characterization package for the declared Byzantine model. At this paper interface, the package gives

$$\mathsf{ByzChar} \iff \mathsf{ByzSafe}. \tag{6.4}$$

with relative maximality for the same characterization relation.

Running-example instantiation. In Section 3, the same trace skeleton is interpreted through profile-side Byzantine obligations when this characterization is enabled.

Series scope note. This theorem is the dynamics-layer Byzantine safety interface. *Harmony from Coherence in Asynchronous MPST* reuses the same bundle and extends it to envelope-level maximality and adherence transport.

Proof sketch. Fix the Byzantine assumption bundle as a typed profile and apply `byzantine_safety_exact_of_profile` to obtain `ExactByzantineSafetyCharacterization` for the profile model.

1. Pin the Byzantine assumption bundle as a typed profile ($\mathsf{H}_{\mathrm{byz}}$components).
2. Project soundness, completeness, and maximality from the extracted exact-characterization object.
3. Interpret soundness and completeness as the two interface implications between ByzCharand ByzSafein this paper's model scope.
4. Record explicit assumption classes for converse sharpness (Corollary 6.4).

The full envelope-maximality development is deferred to *Harmony from Coherence in Asynchronous MPST*. □

Corollary 6.4. Converse Counterexample Families.

Under Assumption Block 6.2, if any required class in $\mathsf{H}_{\mathrm{byz}}$ *is dropped, there exists a counterexample family that violates* ByzSafe. *The dropped classes are quorum or intersection obligations, authentication or evidence-validity obligations, adversarial-budget obligations, and primitive-consistency obligations.*

Proof sketch. Proof mode: witness construction. For each dropped class $\mathcal{A} \in \mathsf{H}_{\mathrm{byz}}$, build witness family $\mathsf{ctr}_{\mathcal{A}}$ with an explicit violated premise map $\delta_{\mathcal{A}} : \mathcal{A} \mapsto \neg\mathsf{Prem}_{\mathcal{A}}$. Intermediate step: execute $\mathsf{ctr}_{\mathcal{A}}$ to produce a safety-visible trace $\tau_{\mathcal{A}}$ with $\mathsf{ByzChar}(\tau_{\mathcal{A}})$ false at the dropped class boundary and $\neg\mathsf{ByzSafe}(\tau_{\mathcal{A}})$. This gives class-indexed converse witnesses and establishes sharpness of Assumption Block 6.2 at this interface scope. □

Proposition 6.5. Byzantine VM-Bridge Interface.
Under Assumption Block 6.2, if theorem-pack capabilities include Byzantine characterization and VM envelope-adherence evidence, then runtime profile claims are constrained by $\mathsf{Eq}^{\mathrm{byz}}_{\mathrm{safe}}$ *under* $\mathcal{E}_{\mathrm{byz}}$*, and claims lacking required capability evidence are rejected by profile admission.*

Proof sketch. Proof mode: compositional bridge. First, `canOperateUnderByzantineEnvelope` validates theorem-pack evidence and yields intermediate gate predicate $\mathsf{GateOK}(p, \Pi)$. Second, profile extraction maps $\mathsf{GateOK}(p, \Pi)$ to envelope-side obligation $\mathsf{ByzEnvOK}(p, \Pi)$. Third, admission/conformance theorems discharge

$$\mathsf{ByzEnvOK}(p, \Pi) \implies \mathsf{Eq}^{\mathrm{byz}}_{\mathrm{safe}} \text{ under } \mathcal{E}_{\mathrm{byz}}.$$

Requests missing required evidence fail at the first step, so interface claims are both executable and proof-carrying. □

VM instruction-layer note. This paper reuses the instruction-form interface and `WellTypedInstr` mapping from the Effect-Typed Bridge in *Coherence for Asynchronous Buffered MPST* (Section 7, Proposition 7.1). At this interface, `send`/`select` carry sender-continuation obligations, `recv`/`branch` carry consume-alignment obligations, and `invoke` carries handler-fragment lift obligations. The contribution here is computable profile extraction and safety-interface packaging on top of that shared VM typing layer.

Table 6.1 summarizes the decidability results.

Predicate	Method	Result
Async subtyping	finite reachable triples	decidable
Regular type equiv.	finite bisim checker	decidable
Crash-stop toler.	residual graph decider	decidable + iff
Byzantine safety	bundle + counterex.	exact iff
Branching feasib.	chromatic criterion	decidable

Table 6.1: Decidability results summary.

Table 6.2 summarizes complexity envelopes for the decision layer.

Predicate	Driver	Complexity
async subtyping	reachable triples	checker: $\mathrm{poly}(N_s + N_e)$
regular type equiv.	finite bisim graph	checker: $\mathrm{poly}(N_s + N_e)$
crash-stop toler.	residual connectivity	checker: $O(N_s + N_e)$
branching feasib.	coloring checks	checker: $\mathrm{poly}(N_s + N_e)$ (fixed profile)

Table 6.2: Complexity envelopes.

Complexity context. For this fragment, any sound complete checker must inspect the explored finite object, so linear graph-traversal in (N_s, N_e) is a baseline lower bound. Table 6.2 reports checker/verifier complexity only; runtime transition-step cost is a separate operational notion handled by the theorem statements, not by these decider envelopes. Witness-bearing checks may add a multiplicative $\mathrm{poly}(W_{\max})$ factor. Outside the regular finite-reachability fragment, known communicating-automata-style analyses can lose decidability (Brand and Zafiropulo, 1983).

7 Quantitative Corollaries

Corollary 7.1. Productive-Step Bound.
For any trace τ *satisfying Assumption Block 4.1,* $P(\tau) \leq W_0$.

Corollary 7.2. Scheduler-Lifted Total Bound (Fairness-Required).

For any trace τ satisfying Assumption Block 4.1 and scheduler profile σ with lift constant κ_σ, $T(\tau) \leq \kappa_\sigma \cdot W_0$.

Fairness-required. Corollary 7.2 depends on the declared scheduler fairness profile through κ_σ.

Corollary 7.3. Critical Capacity Boundary. *Under Assumption Block 4.1, there exists a computable threshold B_c such that the declared phase classifier over buffer capacity is exact at the theorem interface.*

Operational interpretation. $\mathsf{Phase}(B) = \mathsf{Low}$ means capacity is below the certified boundary and profile-side branch obligations are not all realizable. $\mathsf{Phase}(B) = \mathsf{Critical}$ means capacity is exactly at the certified boundary. $\mathsf{Phase}(B) = \mathsf{High}$ means capacity is above the boundary and the same obligations remain realizable with slack.

Proof sketch. Corollary 7.1 is the first clause of Theorem 4.1 with notation specialized to productive-step count. Instantiating Theorem 4.1 at the same trace and premises yields the stated inequality directly. □

Proof sketch. Corollary 7.2 is the scheduler-lift clause of Theorem 4.1 with the same premises and profile constant κ_σ. Substituting the selected scheduler profile into that clause yields the stated bound. □

Proof sketch. `critical_buffer_computable` provides a computable witness value B_c from the capacity predicate over `ConfGraph`. `phase_transition_sharp` proves exact classifier behavior below, at, and above that boundary. Composing these two lemmas yields the stated threshold characterization. □

Table 7.1 separates exact and conservative consequences.

Result	Class
productive-step decrease	exact
productive-step bound by W_0	exact
scheduler-lifted total-step bound	conservative
threshold under stated profile assumptions	exact

Table 7.1: Exact vs conservative results.

Table 7.2 gives worked values used in the worked example.

Quantity	Value
W_0	25
productive-step bound	≤ 25
2-fair total-step bound	≤ 50
B_c	2

Table 7.2: Worked example values.

Equation 7.1 locates the worked example on the phase boundary.

$$\begin{aligned} B < B_c &\Rightarrow \mathsf{Phase}(B) = \mathsf{Low}, \\ B = B_c &\Rightarrow \mathsf{Phase}(B) = \mathsf{Critical}, \\ B > B_c &\Rightarrow \mathsf{Phase}(B) = \mathsf{High}. \\ \text{worked instance: } B_c = 2,\ W_0 = 25,\ B = B_c. \end{aligned} \tag{7.1}$$

With $B_c = 2$ and $W_0 = 25$, the instance sits at the critical capacity line, so productive-step and scheduler-lift conclusions apply exactly as in Theorem 4.1 and Corollaries 7.1–7.2.

8 Proof Architecture

The architecture has one shared operational kernel and two proof branches: the quantitative branch derives global bounds through local-step algebra and strict descent, while the algorithmic branch derives predicate decisions through regularity and finite reachability. This reuse reduces theorem fragmentation and proof duplication. It also gives a direct map from assumptions to output guarantees: Assumption Block 4.1 yields Theorem 4.1 and Corollaries 7.1–7.2; Assumption Block 5.1 yields Theorem 5.3 and the subtyping/equivalence deciders; Assumption Block 6.1 yields Theorem 6.1; and Assumption Block 6.2 yields Theorem 6.3, Corollary 6.4, and Proposition 6.5.

9 Related Work

Classical MPST work established session foundations together with global coherence and projectability discipline (Honda et al., 2008; Honda et al., 2016; Castagna et al., 2012). Projection refinements and local-first compatibility contrasts were

developed in later lines (Majumdar et al., 2021; Scalas and Yoshida, 2018). Logical coherence lines interpret multiparty compatibility as n-ary duality and proof compatibility (Carbone et al., 2015; Carbone et al., 2016; Carbone et al., 2017), and data-certification extensions remained in the same global-projection discipline (Toninho and Yoshida, 2017). Logical formulations connect sessions and propositions and motivate compositional proof interfaces (Caires and Pfenning, 2010, and Wadler, 2012). Program-logical and mechanized lines expanded the verification space (Hinrichsen et al., 2020; Tirore et al., 2025). Event-structure and partial-order approaches give alternate macro views of concurrency (Castellan et al., 2023).

On the quantitative side, our use of descent functions follows the classical drift and stochastic-stability lineage (Lyapunov, 1892; Foster, 1953; Meyn and Tweedie, 2009). On the algorithmic side, the finite-reachability posture follows communicating-automata and model-checking lines (Brand and Zafiropulo, 1983; Baier and Katoen, 2008). Fault-side interfaces are compatible with standard crash and Byzantine theory baselines (Fischer et al., 1985; Chandra and Toueg, 1996; Lamport et al., 1982; Castro and Liskov, 1999).

This paper differs in structure. It unifies quantitative bounds and algorithmic decidability through one regularity kernel. The result is a single reusable theorem program instead of isolated predicate proofs.

10 Limitations and Scope

Quantitative bounds require the Assumption Block 4.1 premises: well-typed asynchronous buffered transitions, active-edge Coherence preservation, non-negativity of W, and an explicit scheduler profile when total-step lifting is claimed. Scheduler-lifted totals remain conservative profile-dependent bounds rather than exact universal runtime counts.

Crash-stop characterization is exact only under Assumption Block 6.1, with crash-stop faults, finite role graph, and residual-graph side conditions. Byzantine results are exact safety characterizations only under Assumption Block 6.2, with $\mathsf{H}_{\mathrm{byz}}$fault-model, authentication and evidence-validity, conflict-exclusion primitive consistency, and adversarial-budget side conditions, and they do not imply Byzantine liveness.

Decidability and finite exploration require the Assumption Block 5.1 regular finite-reachability premises. Non-regular fragments and stronger adversarial models require different constructions.

Reconfiguration commutation and envelope maximality claims are out of scope for this paper and handled separately in the series.

Excluded protocol classes in this paper include non-regular infinite-state families, unguarded recursion patterns that violate finite-reachability criteria, and Byzantine liveness claims under weaker timing assumptions.

Operational failure mode. If regular finite-reachability premises fail, the decision layer may no longer provide terminating/complete checkers (treated as outside decidability scope). If scheduler-profile premises fail, productive-step bounds remain valid but total-step lift claims are withdrawn. If Byzantine profile evidence is missing, capability-gated admission rejects the claim rather than asserting unsafe characterization. Concrete diagnostic. Expected reports are: non-regular fragment rejection at the decider boundary, scheduler-profile deficiency for total-step lift claims, or `canOperateUnderByzantineEnvelope = false` with missing Byzantine bundle evidence classes.

Implementation pointer. Quantitative kernels are in `Runtime/Proofs/WeightedMeasure/*.lean`, decision kernels are in `SessionCoTypes/AsyncSubtyping/*.lean`, and capability-gated Byzantine interfaces are exposed via `Runtime/Proofs/TheoremPack/API.lean`.

11 Conclusion

This paper turns Coherence-preserving dynamics into computable dynamics. Quantitative bounds and decision procedures come from the same regularity source. That result provides the computational bridge between *Coherence for Asynchronous Buffered MPST* and the reconfiguration and envelope claims in *Harmony from Coherence in Asynchronous MPST*.

Byzantine contribution in this paper is the exact safety-characterization interface and converse counterexample family packaging. Byzantine liveness

beyond the stated assumptions remains open and is tracked as future work.

Works Cited

Baier, C., and Katoen, J.-P. (2008). Principles of Model Checking. MIT Press.

Brand, D., and Zafiropulo, P. (1983). On Communicating Finite-State Machines. Journal of the ACM, 30(2), 323–342.

Caires, L., and Pfenning, F. (2010). Session Types as Intuitionistic Linear Propositions. CONCUR 2010.

Carbone, M., Lindley, S., Montesi, F., Schürmann, C., and Wadler, P. (2016). Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types. CONCUR 2016.

Carbone, M., Montesi, F., Schürmann, C., and Yoshida, N. (2015). Multiparty Session Types as Coherence Proofs. CONCUR 2015.

Carbone, M., Montesi, F., Schürmann, C., and Yoshida, N. (2017). Multiparty Session Types as Coherence Proofs. Acta Informatica, 54(3), 243–269.

Castagna, G., Dezani-Ciancaglini, M., Gesbert, N., and Padovani, L. (2012). On Global Types and Multi-Party Sessions.

Castellan, S., et al. (2023). Event-structure and partial-order semantics for session-based concurrency. Journal of Logic and Algebraic Methods in Programming.

Castro, M., and Liskov, B. (1999). Practical Byzantine Fault Tolerance. OSDI 1999.

Chandra, T. D., and Toueg, S. (1996). Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM, 43(2), 225–267.

Cover, T. M., and Thomas, J. A. (2006). Elements of Information Theory (2nd ed.). Wiley.

Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the Presence of Partial Synchrony. Journal of the ACM, 35(2), 288–323.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM, 32(2), 374–382.

Foster, F. G. (1953). On the stochastic matrices associated with certain queueing processes. Annals of Mathematical Statistics, 24(3), 355–360.

Hinrichsen, J., et al. (2020). Actris: Session-type based reasoning in separation logic. POPL 2020.

Honda, K., Yoshida, N., and Carbone, M. (2008). Multiparty Asynchronous Session Types. POPL 2008.

Honda, K., Yoshida, N., and Carbone, M. (2016). Multiparty Asynchronous Session Types. Journal of the ACM, 63(1), Article 9.

Hopcroft, J. E., and Ullman, J. D. (1979). Introduction to Automata Theory, Languages, and Computation. Addison-Wesley.

Karp, R. M., and Miller, R. E. (1969). Parallel Program Schemata. Journal of Computer and System Sciences, 3(2), 147–195.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 4(3), 382–401.

Majumdar, R., Mukund, M., Stutz, F., and Zufferey, D. (2021). Generalising Projection in Asynchronous Multiparty Session Types. CONCUR 2021.

Lynch, N. A. (1996). Distributed Algorithms. Morgan Kaufmann.

Lyapunov, A. M. (1892). The General Problem of the Stability of Motion. Kharkov Mathematical Society.

Meyn, S. P., and Tweedie, R. L. (2009). Markov Chains and Stochastic Stability (2nd ed.). Cambridge University Press.

Scalas, A., and Yoshida, N. (2018). Multiparty Session Types, Beyond Duality. Journal of Logical and Algebraic Methods in Programming, 97, 55–84.

Shannon, C. E. (1948). A Mathematical Theory of Communication. Bell System Technical Journal, 27(3), 379–423 and 27(4), 623–656.

Tirore, L., Bengtson, J., and Carbone, M. (2025). Mechanized MPST metatheory with subject-reduction robustness analysis. ECOOP 2025.

Toninho, B., and Yoshida, N. (2017). Certifying Data in Multiparty Session Types. Journal of Logical and Algebraic Methods in Programming, 90, 61–83.

Wadler, P. (2012). Propositions as Sessions. ICFP 2012.

Appendix

A Deferred Quantitative Proofs

This appendix expands Theorem 4.1 and Corollaries 7.1–7.2.

A.1 Weighted Potential and Productive Steps

For local state s, define:

```
W(s) :=2 * sumDepths(s) + sumBuffers(s)
```

For configuration s, write $W(s)$ for the sum of local/session contributions. A step is productive iff it decreases W strictly.

A.2 Rule-Level Decrease

Lemma A.1. Send/Select Decrease.
For productive send/select steps, $\Delta W \leq -1$.

Proof sketch. Use the coefficient family $W_c = c \cdot \Sigma\mathsf{depth} + \Sigma\mathsf{buffer}$. A productive send or select step consumes one protocol constructor, so depth contributes $-c$. The same step enqueues at most one message, so buffer contributes at most $+1$. Hence $\Delta W_c \leq -c+1$. For $c = 2$, this gives $\Delta W \leq -1$. □

Lemma A.2. Recv/Branch Decrease.
For productive recv/branch steps, $\Delta W \leq -1$.

Proof sketch. For productive recv and branch steps, one buffered head is consumed and no new message is enqueued on that edge. Unfolding the weighted accounting shows a strict negative contribution from the consumed buffer and continuation progress. Therefore $\Delta W \leq -1$. □

Lemma A.3. Configuration Lift.
Rule-level decreases lift to the global step relation:

```
productiveStep s s' → W(s') <= W(s) - 1
```

Proof sketch. Decompose the global step into updated and untouched local components. Apply Lemma A.1 or Lemma A.2 on updated components and use zero change on untouched components. Summing all contributions yields the stated inequality. □

A.3 Trace Bounds

Proposition A.4. Productive-Step Bound.
For any finite trace τ *from* s_0*, productive-step count* $P(\tau)$ *satisfies* $P(\tau) \leq W(s_0)$.

Proof sketch. Index productive positions in the trace and apply Lemma A.3 at each such index. Summing inequalities telescopes to $W(s_0) - W(s_n) \geq P(\tau)$. Since $W(s_n) \geq 0$, we conclude $P(\tau) \leq W(s_0)$. □

Proposition A.5. Scheduler-Lifted Total Bound.
If scheduler profile σ *admits lift constant* κ_σ*, then total steps* T *satisfy:*

$$T \leq \kappa_\sigma \cdot W(s_0). \tag{A.1}$$

Proof sketch. Proposition A.4 bounds the number of productive steps by $W(s_0)$. The scheduler profile contributes a bound on how many non-productive steps can occur between productive ones. Multiplying these bounds gives $T \leq \kappa_\sigma \cdot W(s_0)$. □

Theorem 4.1 and Corollaries 7.1–7.2 are exactly Propositions A.4–A.5 under Assumption Block 4.1.

B Deferred Proof of Algorithmic Decidability Schema

B.1 Abstract Schema

Let $\mathcal{P}$ be a predicate over inputs x. Assume:

1. a finite reachable structure Reach(x),
2. a terminating checker check(x) over Reach(x),
3. soundness/completeness: check(x) = true $\Leftrightarrow$ $\mathcal{P}(x)$.

Theorem B.1. Generic Decider Transfer.
Under the three assumptions above, $\mathcal{P}$ *is decidable.*

Proof sketch. Finiteness of Reach(x) ensures that exploration terminates. The soundness and completeness hypotheses identify check(x) with truth of $\mathcal{P}(x)$. Therefore the checker is a total decision procedure for $\mathcal{P}$. □

B.2 Complexity Envelope (Schema Level)

Let $N_s(x)$ and $N_e(x)$ be explored-state and explored-edge counts for the finite reachable object. If checker runtime is $\mathrm{poly}(N_s(x) + N_e(x))$, then decision complexity is $\mathrm{poly}(N_s(x) + N_e(x))$, optionally multiplied by $\mathrm{poly}(W_{\max})$ when witness objects are validated.

This explains why the paper reports checker complexity in explored-graph size, not in raw syntax size. Runtime transition-step cost is separate.

B.3 Instantiations

1. Async subtyping: reachable triple graph (`reachable_triples_finite`, `async_subtype_decidable`).
2. Regular equivalence: reachable pair/bisim graph (`regular_type_eq_check_sound`, `regular_type_eq_decide_spec`).

C Instantiated Decision Theorems

C.1 Crash-Stop Tolerance Instantiation

Given role graph R and crash set F, let $\mathrm{Residual}(R, F)$ remove crashed roles and incident edges.

Theorem C.1. Crash-Stop Exactness.

$$\mathsf{CrashTolerant}(R, F) \iff \mathsf{ResidualReachable}(R, F). \tag{C.1}$$

Proof sketch. Proof mode: bidirectional reduction. ($\Rightarrow$) If residual reachability fails, choose disconnected components U, V with required endpoint pair $(u, v) \in U \times V$; intermediate cut obligation shows no compliant continuation path, so crash tolerance fails. ($\Leftarrow$) If residual reachability holds, construct continuation witness paths for each required communication edge and compose them into a global crash tolerance witness. Combining both directions yields the equivalence. □

C.2 Branching Feasibility Instantiation

Let `ConfGraph` be the confusability graph induced by branch-distinguishability constraints.

Theorem C.2. Branching Feasibility Criterion.
Feasibility holds iff the declared chromatic-capacity condition on `ConfGraph` holds.

Proof sketch. Proof mode: graph reduction. Encode branch distinguishability obligations as a coloring instance $(\texttt{ConfGraph}, k)$ where k is profile-bounded. Intermediate step: establish translation maps $\mathsf{branchWitness} \leftrightarrow \mathsf{validColoring}$ preserving conflict constraints. Then apply `branching_iff_chromatic_capacity` to conclude feasibility iff the declared chromatic-capacity condition holds. □

C.3 Capacity Threshold Instantiation

Define the phase classifier

$$\mathsf{Phase}(B) = \begin{cases} \mathsf{Low}, & B < B_c, \\ \mathsf{Critical}, & B = B_c, \\ \mathsf{High}, & B > B_c. \end{cases} \tag{C.2}$$

Theorem C.3. Sharp Capacity Boundary.
There exists a computable threshold B_c such that

$$\begin{aligned} \forall B, \quad & (B < B_c \Rightarrow \mathsf{Phase}(B) = \mathsf{Low}) \\ & \wedge (B = B_c \Rightarrow \mathsf{Phase}(B) = \mathsf{Critical}) \\ & \wedge (B > B_c \Rightarrow \mathsf{Phase}(B) = \mathsf{High}). \end{aligned} \tag{C.3}$$

with exactness at the theorem interface.

Proof sketch. `critical_buffer_computable` yields a computable candidate threshold. `phase_transition_sharp` proves that the classifier is exact below, at, and above this threshold. Therefore the phase boundary is both computable and sharp. □

C.4 Byzantine Safety Interface (Scope of This Paper)

Under Assumption Block 6.2, this paper provides the exact safety-side interface and converse counterexample packaging; full envelope maximality and transport are deferred to *Harmony from Coherence in Asynchronous MPST*.

D Exactness and Boundary Notes

D.1 Exact Claims

1. Productive-step decrease and productive-step bounds (Assumption Block 4.1).

2. Critical-capacity boundary characterization (Corollary 7.3) under its stated profile assumptions.
3. Crash-stop characterization (Assumption Block 6.1).
4. Byzantine safety characterization at paper interface scope (Assumption Block 6.2).

D.2 Conservative or Profile-Indexed Claims

1. Scheduler-lifted total-step bounds (κ_σ-indexed).
2. Complexity bounds parameterized by explored finite-state size.

D.3 Out-of-Scope

1. Non-regular fragments for decision transfer.
2. Byzantine liveness under weaker timing/adversary assumptions.
3. Reconfiguration commutation and envelope maximality (*Harmony from Coherence in Asynchronous MPST*).

E Reproducibility

Reproduction workflow and expected outputs are documented in `ARTIFACT.md`. The one-command supplement check is `just artifact-check`. Pinned-commit, DOI, and Lean-statistics rows are synchronized via `bash scripts/paper_repro_rows.sh -sync papers/paper1.tex papers/paper2.tex papers/paper3.tex`. Artifact semantics note. The Lean artifact exposes an explicit trace-level strong fairness predicate (`StrongFairEnv`) refining weak fairness, and the default in-memory runtime transport implements minimal per-edge FIFO enqueue/dequeue behavior.

Artifact Field	**Value**
Repository	`https://github.com/hxrts/telltale`
Pinned commit	`ccddc8ea843684c4 d2d018f673c4861d0664e008`
Archival DOI	`DOI-UNSET`
Lean source statistics	1303 files, 206913 LOC, axioms: 0, unresolved proof holes (`sorry`): 0

Table E.1: Artifact identity and verification statistics for this draft snapshot.

1. Run `just artifact-check`. This runs reproducibility row checks, `just escape`, `just verify-protocols`, paper builds, and manifest generation.
2. Before camera-ready submission, set DOI in `papers/artifact_metadata.env` and run `just paper-repro-check-strict`.

F Anchor Index

Table F.1 maps paper claims to their Lean anchors and source files. Anchors marked with — are infrastructure lemmas used in proofs but not tied to a single claim. Claim entries include assumption-block tags in square brackets. Assumption-tag legend for claim cells: [A1] = Assumption Block 4.1, [A2] = Assumption Block 5.1, [A3] = Assumption Block 6.1, [A4] = Assumption Block 6.2. Functional categories in the table are: Definitions, Theorems, and Runtime Gates. Reading guide for long names: `total_measure_decreasing` is the global productive-descent lift from rule-local ΔW lemmas; `regular_type_eq_decide_spec` states checker iff semantics for regular-type equivalence; `check_rational_effect_witness_complete` is completeness of the rational witness checker for the transport boundary; and `byzantine_safety_exact_of_profile` is the profile-indexed exactness bridge for Byzantine safety claims. Name-pattern note. Parse long anchors as `object_claim_qualifier`: e.g., suffixes like `_of_profile` and `_decide_spec` indicate profile-indexed packaging vs checker-specification theorems.

Anchor	Path (relative to lean/)	Sec.	Claims
Definitions			
weightedMeasure	Runtime/Proofs/WeightedMeasure/Core.lean	§4	—
Theorems			
send_step_decreases	Runtime/Proofs/WeightedMeasure/TotalBound.lean	§4	—
recv_step_decreases	Runtime/Proofs/WeightedMeasure/TotalBound.lean	§4	—
select_step_decreases	Runtime/Proofs/WeightedMeasure/TotalBound.lean	§4	—
branch_step_decreases	Runtime/Proofs/WeightedMeasure/TotalBound.lean	§4	—
total_measure_decreasing	Runtime/Proofs/WeightedMeasure/TotalBound.lean	§4	Thm. 4.1, Cors. 7.1–7.2 [A1]
reachable_triples_finite	SessionCoTypes/AsyncSubtyping/Core.lean	§5	—
async_subtype_decidable	SessionCoTypes/AsyncSubtyping/Core.lean	§5	Thm. 5.3 [A2]
reachable_pairs_finite	SessionCoTypes/Coinductive/BisimDecidable/Correctness.lean	§5	—
regular_type_eq_check_sound	SessionCoTypes/Coinductive/BisimDecidable/Decidable.lean	§5	—
regular_type_eq_decide_spec	SessionCoTypes/Coinductive/BisimDecidable/Decidable.lean	§5	Thm. 5.3 [A2]
rational_effect_transport_bridge	Runtime/Proofs/EffectBisim/RationalFragment.lean	§5	—
check_rational_effect_witness_sound	Runtime/Proofs/EffectBisim/RationalFragment.lean	§5	—
check_rational_effect_witness_complete	Runtime/Proofs/EffectBisim/RationalFragment.lean	§5	—
strict_boundary_witness_effect	Runtime/Proofs/EffectBisim/RationalFragment.lean	§5	—
crash_tolerant_dec_sound	Protocol/CrashTolerance.lean	§6	—
crash_tolerance_iff	Protocol/CrashTolerance.lean	§6	Thm. 6.1 [A3]
byzantine_safety_exact_of_profile	Runtime/Proofs/Adapters/Distributed/EnvelopeTheorems.lean	§6	Thm. 6.3, Cor. 6.4 [A4]
branching_iff_chromatic_capacity	Protocol/SpatialBranching.lean	§C	Thm. C.2
critical_buffer_computable	Protocol/BufferBoundedness/PhaseSharpness.lean	§7	—
phase_transition_sharp	Protocol/BufferBoundedness/PhaseSharpness.lean	§7	Cor. 7.3 [A1]
Runtime Gates			
canOperateUnderByzantineEnvelope	Runtime/Proofs/TheoremPack/API.lean	§6	Prop. 6.5 [A4]

Table F.1: Anchor index.