Chapter 2: On Monitoring Compliance for Open-Ended Two Agent Contracts

kharrazkaram

September 2025

Contents

1 Overview

1.1 Motivation

Normative specifications are often presented as constraints on what an agent must do or must avoid.

In many practical contexts, however, compliance is not achieved solely by individual agents.

Instead, compliance results from interactions in which each party may enable, disregard, or obstruct the other party's actions.

This chapter examines collaborative normative settings in which agents are not engaged in zero-sum games, yet strategic behavior and conflicting incentives may still arise.

The analysis focuses on two agents, as the fundamental challenges are already present in the simplest bilateral contracts.

Interactions and collaboration. The standard deontic formalism describes the duties and prohibitions of a single agent in isolation.

In contrast, many norms require that a desired outcome be realized only through the contributions of both parties.

For instance, a tenant may attempt to pay rent, but the payment may succeed only if the landlord provides a payment channel or accepts the transfer.

Similarly, occupancy is meaningful only if one party grants access while the other actually occupies.

A robust model must therefore distinguish between (i) attempts made by each agent and (ii) successful collaboration that results from a compatible combination of these attempts.

Additionally, the model must account for non-interference, in which compliance requires one party to refrain from actions that would prevent the other from fulfilling its obligations.

Open-ended normative interactions. Normative specifications may govern long-term collaborations.

For instance, contracts typically govern interactions that can persist indefinitely unless properly terminated, often with a periodic structure.

A rental agreement, for example, imposes recurring monthly obligations, conditional repairs triggered by requests, and contrary-to-duty obligations that arise after non-compliance.

This motivates the development of a trace model that supports (a) aggregation at the level specified by the norm (e.g., months) and (b) prefix-based evaluation, as monitoring and dispute resolution depend on partial observations of ongoing relationships.

Blame and accountability. When a collaborative norm is breached, it is insufficient to merely state that the contract has been violated.

It is necessary to identify who is responsible for the violation and to provide justification for this attribution.

This requirement is equally important in contractual disputes and regulatory contexts, where enforcement decisions depend on explicit accounts of causation and justification.

In bilateral scenarios such as landlord–tenant disputes, courts typically reconstruct a timeline to determine which party obstructed the other: for example, whether the tenant failed to pay while entitled to possession, whether the landlord denied access without adhering to legal procedures (such as notice, deadlines, or court orders), or whether the landlord's denial of access justified rent withholding or reduction. Frequently, both parties contribute to escalation, and resolution depends on allocating remedies proportionate to the harms caused at each stage, including restoring access, ordering payment of arrears, granting rent reductions for unusable periods, awarding damages for proven losses, and verifying compliance with termination and eviction procedures. In addition to qualitative attribution, practical compliance analysis requires a quantitative assessment of the extent of non-compliance, such as the number of months' rent unpaid, frequency of denied access, or missed repair obligations. A blame concept suitable for compliance tools must therefore be formally defined, operationally verifiable, and robust under incremental observation. This chapter develops such a notion, implements executable monitors, and extends the framework with quantitative measures that surpass the traditional first-failure perspective.

Research questions. These considerations motivate the following research questions from a formal method perspective:

1. How can collaboration be modeled as a first-class semantic object, distinguishing individual attempts from jointly successful outcomes?

2. Which trace abstractions effectively capture both cooperative behavior and non-interference at the granularity defined by contractual periods?

3. How can open-ended, potentially infinite, streams of duties and rights be specified and monitored within a contractual framework?

4. How can blame and quantitative accountability be defined to more accurately reflect real-world legal reasoning and support automated monitoring?

1.2 Methodology

To address the research questions, we adopt a *constructive formal methods* approach. Our methodology is predicated on the rigorous separation of *specification* (the denotational meaning of a contract) from *implementation* (the operational mechanism of monitoring). This separation ensures that the resulting tools are correct-by-construction rather than ad-hoc scripts.

The research follows a four-phase structural pipeline:

Phase 1: Semantic Abstraction. We begin by abstracting the domain of interaction. Methodologically, we reject standard atomic trace models in favor of a *multi-layered* approach. We model interaction not as a given sequence of events, but as the semantic intersection of independent agent strategies. This allows us to reason formally about causal concepts like obstruction and interference before defining normative rules.

Phase 2: Denotational Specification. We adopt a *semantics-first* strategy. Rather than defining the logic via its execution (e.g., how an automaton processes it), we first define its abstract truth conditions. Our approach prioritizes *tightness*, the ability to render progressive and precise verdicts on finite prefixes and distinguish exact (tight) prefixes for violation/satisfaction from their equivalent post continuation. We start with a binary compliance semantics, refine it into a blame assignment model, and finally extend it to a quantitative domain. This layering ensures that complex accountability metrics remain consistent with the fundamental notion of compliance.

Phase 3: Structural Synthesis. To bridge theory and practice, we employ *automata-theoretic synthesis*. We treat the logic as a blueprint for generating deterministic Moore machines. Our methodology relies on structural induction: we define an operational transformation for every logical operator, ensuring that the structure of the monitor mirrors the structure of the contract.

Phase 4: Formal Verification. Finally, we validate the pipeline through *conformance proofs*. We do not merely test the monitors; we prove that the synthesized automata are logically isomorphic to the denotational semantics defined in Phase 2. This closes the loop, guaranteeing that the executable artifacts faithfully represent the normative intent.

1.3 Contributions

This chapter establishes a comprehensive framework for the specification and runtime verification of collaborative normative systems. The primary contribution is a fully verified pipeline bridging high-level normative abstractions and low-level executable monitors.

The specific theoretical and technical contributions are:

(Con1) **Attempt-Based Interaction Model.** We introduce a formal algebra that distinguishes between an agent's *attempt* and the *interaction outcome*. Unlike standard models that conflate "failure to act" with "blocked action," our model treats non-interference as a first-class semantic property. This enables the formal verification of "active participation"—determining if an agent did everything in their power to fulfill a duty.

(Con2) **The Two-Agents Collaborative Normative Logic** (TACNL). We formalize TACNL , a logic tailored for bilateral, periodic contracts over atomic actions. It extends standard deontic specifications with:

- *Atomic Responsibility and Power Modalities:* Explicit constructs for Obligation, Prohibition, and Power, parameterized by the responsible/beneficiary agent, and the target action.
- *Regular Expression Control Flow:* The use of regular expressions to define complex temporal *triggers* for activating duties and *guards* for termination conditions.
- *Contrary-to-duty operators* that handle reparable violations.
- *Explicit Repetition Operator:* A dedicated operator for modeling periodic, recurring obligations (e.g., ”pay rent every month”). Unlike Linear Temporal Logic (LTL), which typically encodes repetition through nested ”Always” ($\Box$) and ”Next” ($\bigcirc$) operators that flatten the temporal structure, our repetition operator preserves the *periodicity* as a first-class structural element.

Crucially, TACNL is defined over the attempt-based model, allowing it to capture the collaborative nuances of real-world contracts.

(Con3) **Theory of Tight and Quantitative Semantics.** We introduce a family of *Forward Tight Semantics* designed for online monitoring.

- We define a **5-valued verdict domain** that distinguishes between eager (first) violations and irrelevant post-violation extensions.
- We extend this to a **Quantitative Blame Semantics** that accumulates violations over time, allowing for the calculation of cumulative penalties in open-ended contracts.

(Con4) **Correct-by-Construction Monitor Synthesis.** We provide a compilation algorithm that transforms any TACNL specification into a deterministic Moore machine. We provide structural proofs demonstrating that this monitor is *operationally equivalent* to the tight semantics. This result eliminates the need to verify the monitoring tool itself, as correctness is guaranteed by the synthesis process.

(Con5) **Formalization of Accountability.** Beyond binary compliance, we contribute a formal definition of *blame assignment* in cooperative settings. We prove that our blame semantics is a consistency-preserving refinement of the standard violation semantics, providing a mathematical basis for dispute resolution and liability assessment.

Organization. We first ground the problem in a rental-agreement example and extract the modeling requirements.

We then develop the trace and synchronization layer for collaborative interaction, introduce TACNL , and finally present the monitor-based semantics, blame attribution, and quantitative extensions.

2 Motivating Example

To ground the subsequent theoretical framework in a concrete setting, we select a representative scenario from the domain of property law: a residential **flat rental agreement**. We deliberately choose a simplified version of such a contract for the sake of clarity and pedagogical effectiveness. By stripping away the complex boilerplate typical of real-world jurisprudence, we obtain a accessible structure that isolates the core logical phenomena we wish to study—namely, the collaborative dynamics between agent attempts and the temporal evolution of duties. This abstraction allows the reader to focus on the semantic challenges of monitoring without being overwhelmed by extraneous legal details.

Example 1. *We present a simplified rental contract between a tenant (agent 1) and a landlord (agent 2). The clauses illustrate cooperative actions such as paying rent and granting occupancy: payment requires both the tenant's offer and the landlord's acceptance, while occupancy requires both the tenant's willingness to occupy and the landlord's permission. The contract also includes a reparation clause and a termination condition. All clauses are monthly regulated and repeat over time.*

Occupancy and Rent Payment

(C1) The Tenant shall pay the full monthly rent on or before the due date.

(C2) The Landlord shall guarantee the Tenant's right to quiet enjoyment and occupancy of the premises, provided the Tenant complies with the agreement.

(C3) In the event of non-payment or late payment of rent:

- *(i) If the Tenant fails to pay by the due date, a late fee of 10*
- *(ii) This late fee shall be due and payable along with the next month's rent.*
- *(iii) The late fee does not waive any other rights or remedies available to the Landlord under this agreement or the law.*

(C4) If the Tenant submits a formal request for necessary repairs, the Landlord shall carry out the required repairs within one (1) month of receiving the request.

Termination Notice and Continued Occupancy

(C5) The Tenant may terminate this rental agreement by giving written notice at least three (3) months prior to the intended termination date. Upon doing so, the Tenant shall:

- *(i) Continue to pay the full monthly rent during the three-month notice period.*
- *(ii) Retain the right to occupy the premises for the entirety of the notice period.*
- *(iii) Comply with all other terms of the rental agreement during the notice period.*
- *(iv) Vacate the premises no later than the final day of the notice period, unless otherwise agreed in writing.*

Informal Projecton of Interaction Scenarios on the Contract

To ground these challenges in concrete behavior, let us consider three hypothetical execution traces (scenarios) over a timeline of months $t = 1, 2, \ldots$. These scenarios highlight the divergence between simple observation (what happened) and normative analysis (who is responsible).

Scenario 1: Obstruction (The "Handshake" Problem). Consider the obligation to pay rent (Clause C1) at month $t = 1$.

- **Action:** The Tenant attempts to initiate the bank transfer.
- **Reaction:** The Landlord, wishing to manufacture a reason for eviction, closes the receiving bank account or refuses the check.
- **Observable Outcome:** No money is transferred.
- **Normative Analysis:** A naive monitor observing only "payment status" would flag a violation by the Tenant. However, in our collaborative framework, the Tenant has fulfilled their duty by *attempting* the act. The failure is attributed to the Landlord's interference. This necessitates distinguishing *attempted payment* from *successful payment*.

Scenario 2: Violation and Reparation (The CTD Chain). Consider a scenario where the Tenant genuinely runs out of funds.

- **Month 1:** Tenant fails to pay rent.
- **Immediate Verdict:** Violation of C1. The contract enters a "violated" state.
- **Normative Shift:** Clause C3 (Reparation) activates. The Tenant now holds a *secondary obligation*: pay Month 1 Rent + Month 2 Rent + 10% Fee at Month 2.
- **Month 2:** Tenant pays the total accumulated amount.
- **Final Verdict:** The violation is considered *repaired*. The contract returns to a compliant state. The monitor must track this "state recovery" rather than permanently aborting at the first failure.

Scenario 3: Termination and Temporal Reconfiguration. Consider the Tenant exercising the power to terminate (Clause C5).

- **Month 5:** Tenant issues a formal written notice to terminate.
- **Normative Shift:** This is not a violation, but a *state change*. The contract, previously open-ended, is now fixed to terminate at Month 8 $(5+3)$.
- **Month 6–8:** The Tenant stops paying rent, assuming the notice ended the contract immediately.

- **Verdict:** Violation. The monitor must enforce that the *obligations persist* during the notice period (Clause C5-i). The exercising of power did not delete the obligations but merely set a horizon for their expiration.

Formal Modeling Challenges

While conceptually straightforward, formalizing this contract requires resolving several semantic challenges that standard specification languages (such as LTL or standard Deontic Logic) fail to address adequately.

1. The "Handshake" Semantics (Attempts vs. Outcomes). Clauses C1 and C2 illustrate that compliance is a *joint achievement*. For example, "Rent Payment" (C1) semantically implies that the Tenant offers the money *and* the Landlord accepts it. Standard trace models often represent this as a single atomic event *pay*. However, in a dispute, we must distinguish between:

- The Tenant failing to offer rent (Tenant violation).
- The Tenant offering, but the Landlord blocking the transfer (Landlord interference).

Challenge: The formalism must treat *attempts* and *successful outcomes* as distinct semantic layers to correctly attribute blame.

2. Contrary-to-Duty Structures (Reparation). Clause C3 introduces non-monotonicity. If C1 is violated, the contract does not simply evaluate to *false* (breach). Instead, it transitions to a sub-ideal state where a secondary obligation (the late fee) becomes active. This creates a *reparation chain*: Primary Duty $\rightarrow$ Violation $\rightarrow$ Secondary Duty. **Challenge:** The logic must support *recoverable violations*, allowing the monitoring status to fluctuate between "violated" and "repaired" over time.

3. Reactive Triggers and Monitoring Horizons. Clause C4 (Repairs) imposes a duty with a relative deadline: "within one month of request." From a monitoring perspective, this requires *forward-looking tightness*. We cannot wait for an infinite trace to determine if the repair happened. **Challenge:** The semantics must identify the *exact finite prefix* where the deadline is missed to trigger a verdict immediately (eager rejection).

4. Hohfeldian Powers and Structural Change. Clause C5 (Termination) represents a *Power* rather than a simple duty. By exercising this power, the Tenant unilaterally alters the normative landscape, changing an open-ended/infinite contract into a finite one (ending in 3 months). **Challenge:** The framework must model agents not only as actors within the rules but as entities capable of modifying the rules (or their duration) dynamically.

Summary

This motivates the need for a new logic of collaboration, one that can:

1. represents the strategies of multiple agents acting in tandem rather than in opposition,
2. expresses obligations, powers, reparations, and triggers in a uniform framework,
3. capture open-ended dynamics where obligations evolve over time, and
4. supports the analysis of compliance, blame, and verification of strategies against contract clauses.

Such a framework requires both a formal model of interacting agent behaviors and a language with syntax and semantics suited to normative reasoning in collaborative, time-sensitive contracts.

The subsequent section introduces a formal model of interaction. We define the collaborative outcome as a trace derived from the synchronization of two independent traces, each capturing the behavior of a distinct agent.

3 Trace Synchronization in Multi-Agent Models

Why synchronization? In single-agent modal and temporal logic, one evaluates formulas over a single run or Kripke path. In multi-agent systems, each agent evolves according to its own local transition structure. A global execution must therefore align these local evaluations into a coherent trace. The alignment policy is the *synchronization discipline*. Different communities have standardized different choices: pure interleaving from process calculi [15, 12], true-concurrency and trace theory to separate independence from conflict [6], round-based lockstep in synchronous languages [10], and clock-synchronous yet action-asynchronous products in networks of timed automata [2]. Strategic and epistemic formalisms build on the same global-trace view: ATL interprets temporal modalities over outcomes of joint strategies [3], and epistemic modal operators quantify over indistinguishable classes of such global traces [7]. The choice of synchronization is not cosmetic. It clarifies what constitutes feasible joint behavior, which, in turn, drives semantic validity, strategy quantification, and downstream analysis of compliance, responsibility, or blame.

Time assumptions and word types. Synchronization disciplines adopt different assumptions about time. Interleaving and handshake reason about order without a global clock. Lockstep assumes a single round clock. Timed models use absolute time stamps. We work with two agents $\mathsf{agt} = \{1,2\}$, discrete time $\mathbb{N}$, and per-agent alphabets Σ_i.

Motivation. We introduce the attempted/successful abstraction, suited for capturing interference and collaboration between agents in the context of collaborative normative specifications, and discuss how it resembles and differs from existing synchronization techniques in the literature. This contribution is not cosmetic: It determines which joint behaviors are feasible and, therefore, what formulas are satisfied, and how knowledge and responsibility are assessed.

Method and structure. We proceed in three steps. (1) *Action–centric models.* We introduce the three standard trace notions: logical-time words (order only), metric-timed words (labels with time stamps), and synchronous-time words (round-indexed with explicit stutter). Then we connect the models: From a timed word, we define two morphisms: LT (drops time stamps, preserves order) and ST (pads to a global clock), with formal properties and examples, highlighting what is preserved and what is forgotten. (2) *Synchronization operators.* We explore the common synchronization operators in the literature : the asynchronous interleaving $|||$ (all order-preserving shuffles), the lockstep zip $||_{\text{lock}}$ (round-by-round alignment), and the lockstep-with-handshakes $||_{\text{hs}}^{A}$ (joint actions constrained on A), each with definitions, intended reading, and worked examples. (3) *attempted, prevented, and successful collaboration.* We introduce our contribution of attempted/successful abstraction on periodic, agent-tagged set traces: (i) *attempted* collaboration is captured by per-agent enabler sets (active participation) in a period; (ii) *prevented* collaboration arises from blocker (non-interference violations); and (iii) *successful* collaboration is defined by the operator Succ, the point-wise intersection of the two agents' untagged suggested sets (equivalently, the set-theoretic image of lockstep-with-handshakes on the collaboration alphabet).

3.1 The State of the Art Action-Centric Trace Models

Before studying synchronization mechanisms, we first fix *action-only* models of agent behavior and the canonical morphisms between them. We work with two agents $\mathsf{agt} = \{1,2\}$, per-agent alphabets Σ_i, and discrete time $\mathbb{N}$.

3.1.1 Logical-Time Words

Logical-time (untimed) traces capture *order only*, abstracting away any notion of time. They are the staple semantic objects in interleaving models of process calculi (CCS/CSP) and in Mazurkiewicz trace theory for true concurrency [15, 12, 6].

Definition 1 (Logical-time word, notation τ_i^{lt})**.** *A (finite) logical-time word of agent i over an alphabet Σ_i is a sequence:*

$$\tau_i^{\text{lt}} \;=\; \langle (a_0^i), (a_1^i), \ldots, (a_{n_i-1}^i) \rangle \;\in\; \Sigma_i^*,$$

where $a_k^i \in \Sigma_i$ for all k. Size (length)*:* $|\tau_i^{\text{lt}}| := n_i$. Positions*:* $\mathsf{pos}(\tau_i^{\text{lt}}) := \{0,1,\ldots,n_i - 1\}$. Indexing*:* $\tau_i^{\text{lt}}[k] := a_k^i$ *for $k \in \mathsf{pos}(\tau_i^{\text{lt}})$. We write $\langle - \rangle$ for the empty word and "$\cdot$" for concatenation.*

Basic operators and short hands. For $A \subseteq \Sigma_i$, the projection $\tau_i^{\text{lt}}\lceil A \in A^*$ deletes all symbols not in A. For $a \in \Sigma_i$, $\#_a(\tau_i^{\text{lt}}) := |\{\, k \in \mathsf{pos}(\tau_i^{\text{lt}}) \mid \tau_i^{\text{lt}}[k] = a \,\}|$. Prefixes by length: $\tau_i^{\text{lt}}[0..k] := \langle (a_0^i), \ldots, (a_k^i) \rangle$ for $0 \le k < n_i$.

Example 2 (Two agents, order-only — showing all operators)**.** *Let* $\Sigma_1 = \{\mathit{pick}, \mathit{handover}\}$ *and* $\Sigma_2 = \{\mathit{scan}, \mathit{handover}\}$*. Take*

$$\tau_1^{\mathrm{lt}} = \langle(\mathit{pick}), (\mathit{handover})\rangle, \qquad \tau_2^{\mathrm{lt}} = \langle(\mathit{scan}), (\mathit{handover})\rangle.$$

Size/positions/indexing: $|\tau_1^{\mathrm{lt}}| = |\tau_2^{\mathrm{lt}}| = 2$, $\mathsf{pos}(\tau_1^{\mathrm{lt}}) = \{0,1\}$, $\tau_1^{\mathrm{lt}}[0] = \mathit{pick}$, $\tau_1^{\mathrm{lt}}[1] = \mathit{handover}$.
Counts: $\#_{\mathit{handover}}(\tau_1^{\mathrm{lt}}) = \#_{\mathit{handover}}(\tau_2^{\mathrm{lt}}) = 1$
Projection: *with* $A = \{\mathit{handover}\}$, $\tau_1^{\mathrm{lt}}{\restriction}A = \langle(\mathit{handover})\rangle$. Prefix: $\tau_1^{\mathrm{lt}}[0..0] = \langle(\mathit{pick})\rangle$.
Concatenation: $\tau_1^{\mathrm{lt}} \cdot \langle(\mathit{handover})\rangle = \langle(\mathit{pick}), (\mathit{handover}), (\mathit{handover})\rangle$.

3.1.2 Metric-Timed Words

Context. Metric-timed traces record *which* action occurs and *when* it occurs. They are standard in timed automata and timed process theories [2].

Definition 2 (Timed word, notation τ_i^{mt})**.** *Fix agent* $i \in \{1,2\}$ *with alphabet* Σ_i *and discrete time* $\mathbb{N}$*. A (finite) timed word is*

$$\tau_i^{\mathrm{mt}} = \langle(a_0^i, t_0^i), (a_1^i, t_1^i), \ldots, (a_{n_i-1}^i, t_{n_i-1}^i)\rangle \in (\Sigma_i \times \mathbb{N})^*,$$

with $a_k^i \in \Sigma_i$, $t_k^i \in \mathbb{N}$, *and* $0 \le t_0^i < \cdots < t_{n_i-1}^i$. Size*:* $|\tau_i^{\mathrm{mt}}| := n_i$. Positions*:* $\mathsf{pos}(\tau_i^{\mathrm{mt}}) := \{0,1,\ldots,n_i-1\}$. Indexing by position*:* $\tau_i^{\mathrm{mt}}[k] := (a_k^i, t_k^i)$*. Accessors:* $\mathsf{lab}(\tau_i^{\mathrm{mt}}, k) := a_k^i$, $\mathsf{ts}(\tau_i^{\mathrm{mt}}, k) := t_k^i$. Time set/horizon*:* $\mathsf{time}(\tau_i^{\mathrm{mt}}) := \{t_k^i\}$, $\mathsf{last}(\tau_i^{\mathrm{mt}}) := t_{n_i-1}^i$ *(if* $n_i > 0$*).* Span*:* $\mathsf{span}(\tau_i^{\mathrm{mt}}) := t_{n_i-1}^i - t_0^i$ *(if* $n_i > 0$*).* Indexing by time (partial)*:* $\rho_i(t) = a$ *iff* (a,t) *occurs in* τ_i^{mt}*; else undefined. We write* $\langle - \rangle$ *for the empty timed word; "$\cdot$" is concatenation.*

Basic operators and short hands. For $A \subseteq \Sigma_i$, projection $\tau_i^{\mathrm{mt}}{\restriction}A \in (A \times \mathbb{N})^*$ keeps exactly (a,t) with $a \in A$. Prefixes: by *length* $\tau_i^{\mathrm{mt}}[0..k]$, and by *time* $\tau_i^{\mathrm{mt}}{\restriction}\{t \le T\}$ (shorthand: $\tau_i^{\mathrm{mt}}\{t \le T\}$). Inter-event gaps: $\Delta_k := t_{k+1}^i - t_k^i > 0$.

Example 3 (Two agents with time stamps — showing all operators)**.** *Badge reader (agent 1) and back end (agent 2):*

$$\Sigma_1 = \{\mathit{badge\_tapped}, \mathit{door\_open}\}, \quad \Sigma_2 = \{\mathit{auth\_ok}\}.$$

Run:

$$\tau_1^{\mathrm{mt}} = \langle(\mathit{badge\_tapped}, 1), (\mathit{door\_open}, 3)\rangle, \qquad \tau_2^{\mathrm{mt}} = \langle(\mathit{auth\_ok}, 2)\rangle.$$

Size/indexing: $|\tau_1^{\mathrm{mt}}| = 2$, $\mathsf{pos}(\tau_1^{\mathrm{mt}}) = \{0,1\}$, $\tau_1^{\mathrm{mt}}[0] = (\mathit{badge\_tapped}, 1)$, $\mathsf{lab}(\tau_1^{\mathrm{mt}}, 1) = \mathit{door\_open}$, $\mathsf{ts}(\tau_1^{\mathrm{mt}}, 1) = 3$.
Time set/horizon/span: $\mathsf{time}(\tau_1^{\mathrm{mt}}) = \{1,3\}$, $\mathsf{last}(\tau_1^{\mathrm{mt}}) = 3$, $\mathsf{span}(\tau_1^{\mathrm{mt}}) = 3 - 1 = 2$.
Indexing by time: $\rho_1(3) = \mathit{door\_open}$, $\rho_1(2)$ *undefined.*
Projection: *with* $A = \{\mathit{door\_open}\}$, $\tau_1^{\mathrm{mt}}{\restriction}A = \langle(\mathit{door\_open}, 3)\rangle$.
Length-prefix: $\tau_1^{\mathrm{mt}}[0..0] = \langle(\mathit{badge\_tapped}, 1)\rangle$.
Time-prefix: $\tau_1^{\mathrm{mt}}\{t \le 2\} = \langle(\mathit{badge\_tapped}, 1)\rangle$, $\quad \tau_1^{\mathrm{mt}}\{t \le 3\} = \tau_1^{\mathrm{mt}}$.
Gaps: $\Delta_0 = 3 - 1 = 2$.
Concatenation: $\tau_1^{\mathrm{mt}} \cdot \langle(\mathit{audit\_log}, 4)\rangle = \langle(\mathit{badge\_tapped}, 1), (\mathit{door\_open}, 3), (\mathit{audit\_log}, 4)\rangle$.

3.1.3 Synchronous-Time Words

Synchronous-time traces assume a *single global logical clock* (the *synchronous hypothesis*): Each tick is an instantaneous round in which components react simultaneously; the absence of an action at a tick is explicit. This is the semantic backbone of synchronous languages such as Lustre and Esterel [10].

Definition 3 (Synchronous-time word, notation τ_i^{st})**.** *Fix agent $i \in \{1,2\}$ with alphabet Σ_i and a stutter symbol "$-$" with $- \notin \Sigma_i$. A (finite) synchronous-time word is*

$$\tau_i^{\mathrm{st}} \;=\; \langle (s_0), (s_1), \ldots, (s_T) \rangle \;\in\; (\Sigma_i \cup \{-\})^*,$$

with $s_t \in \Sigma_i \cup \{-\}$ the observation at round t. Size*: $|\tau_i^{\mathrm{st}}| := T+1$ if nonempty (else 0).* Positions*: $\mathsf{pos}(\tau_i^{\mathrm{st}}) := \{0,1,\ldots,T\}$.* Indexing*: $\tau_i^{\mathrm{st}}[t] := s_t$. T is the* horizon*.*

Basic operators and short hands. For $A \subseteq \Sigma_i$:

- *Stutter-preserving projection* $\tau_i^{\mathrm{st}}\lceil A \in (A \cup \{-\})^*$ keeps letters in A and all "$-$".
- *Logical (stutter-erasing) projection* $\mathsf{erase}_{-}(\tau_i^{\mathrm{st}}\lceil A) \in A^*$ drops all "$-$".

For $a \in \Sigma_i$:

$$\#_a(\tau_i^{\mathrm{st}}) := |\{t \in \mathsf{pos}(\tau_i^{\mathrm{st}}) \mid \tau_i^{\mathrm{st}}[t] = a\}|, \qquad \mathsf{act}(\tau_i^{\mathrm{st}}) := \{t \mid \tau_i^{\mathrm{st}}[t] \neq -\}.$$

Round-prefix $\tau_i^{\mathrm{st}}[0..r] := \langle (s_0), \ldots, (s_r) \rangle$ for $0 \le r \le T$.

Example 4 (Two controllers at 1 Hz — illustrating all operators)**.** *Consider HVAC (agent 1) and Security (agent 2) sharing a global clock:*

$$\Sigma_1 = \{\mathit{heat\_on}\}, \qquad \Sigma_2 = \{\mathit{door\_lock}\}.$$

In one episode, we record

$$\tau_1^{\mathrm{st}} = \langle (-), (-), (\mathit{heat\_on}) \rangle, \qquad \tau_2^{\mathrm{st}} = \langle (-), (\mathit{door\_lock}), (-) \rangle.$$

Size, positions, indexing:
$|\tau_1^{\mathrm{st}}| = |\tau_2^{\mathrm{st}}| = 3$ *with horizon $T = 2$,* $\mathsf{pos}(\tau_1^{\mathrm{st}}) = \{0,1,2\}$*,* $\tau_1^{\mathrm{st}}[2] = \mathit{heat\_on}$*.*
Counts and active rounds:
$\#_{\mathit{heat\_on}}(\tau_1^{\mathrm{st}}) = 1$*,* $\mathsf{act}(\tau_1^{\mathrm{st}}) = \{2\}$*.*
Stutter-preserving projection:
with $A = \emptyset$, $\tau_1^{\mathrm{st}}\lceil A = \langle (-), (-), (-) \rangle$*.*
Logical (stutter-erasing) projection:
$\mathsf{erase}_{-}(\tau_1^{\mathrm{st}}\lceil A) = \langle - \rangle$*, while* $\mathsf{erase}_{-}(\tau_1^{\mathrm{st}}) = \langle (\mathit{heat\_on}) \rangle$*.*
Round-prefix:
$\tau_2^{\mathrm{st}}[0..1] = \langle (-), (\mathit{door\_lock}) \rangle$*.*
Concatenation:
$\tau_2^{\mathrm{st}} \cdot \langle (-) \rangle = \langle (-), (\mathit{door\_lock}), (-), (-) \rangle$*.*

3.1.4 Connecting the Models

The objective of this section is to connect the three action-centric trace models introduced above. Given a metric-timed word τ_i^{mt}, we introduce two canonical morphisms that deliberately discard part of the information: the *logical-time projection* LT, which erases time stamps while preserving the order of events, and the *synchronous padding* ST, which aligns events to a global round clock by inserting explicit stutter symbols "$-$". We assume throughout that time is discrete and that one tick of the synchronous clock corresponds to a single unit of metric time.

From metric time to logical time. The map LT drops time stamps but preserves labels and their order. It is the coarsest view that still distinguishes different event sequences.

Example 5 (Motivation for LT: order-only view)**.**

$$\begin{aligned}&\textit{Let } \tau_i^{\mathrm{mt}} = \langle(\mathit{login},1),(\mathit{auth\_ok},4),(\mathit{door\_open},5)\rangle,\\ &\textit{Then } \mathsf{LT}(\tau_i^{\mathrm{mt}}) = \langle(\mathit{login}),(\mathit{auth\_ok}),(\mathit{door\_open})\rangle.\end{aligned}$$

With $|\mathsf{LT}(\tau_i^{\mathrm{mt}})| = |\tau_i^{\mathrm{mt}}| = 3$.
Note that if we "retime" the same events to $\langle(\mathit{login},10),(\mathit{auth\_ok},11),(\mathit{door\_open},100)\rangle$*,*
The LT*-image will be the same.*

Lemma 1 (Logical-time projection LT: order-only view)**.** *Let* $\tau_i^{\mathrm{mt}} = \langle(a_0^i,t_0^i),\dots,(a_{n_i-1}^i,t_{n_i-1}^i)\rangle$ *be a timed word. There exists a unique logical word* $\tau_i^{\mathrm{lt}} = \mathsf{LT}(\tau_i^{\mathrm{mt}}) \in \Sigma_i^*$ *satisfying:*

- ***Number of event preservation***

$$|\tau_i^{\mathrm{lt}}| \;=\; |\tau_i^{\mathrm{mt}}| \;=\; n_i.$$

- ***Action occurrence preservation***

$$\forall a \in \Sigma_i : \quad \#_a\big(\tau_i^{\mathrm{lt}}\big) \;=\; \#_a\big(\tau_i^{\mathrm{mt}}\big).$$

- ***Event order preservation***

$$\tau_i^{\mathrm{mt}}[k] \textit{ precedes } \tau_i^{\mathrm{mt}}[\ell] \;\Rightarrow\; \tau_i^{\mathrm{lt}}[k] \textit{ precedes } \tau_i^{\mathrm{lt}}[\ell] \textit{ in } \tau_i^{\mathrm{lt}}.$$

- ***Per-action order preservation***

$$a_k^i = a_\ell^i \wedge k < \ell \;\Rightarrow\; \textit{the k-th a precedes the } \ell\textit{-th a in } \tau_i^{\mathrm{lt}}.$$

- ***Projection compatibility***

$$\forall A \subseteq \Sigma_i : \qquad \big(\mathsf{LT}(\tau_i^{\mathrm{mt}})\big)\!\restriction\! A \;=\; \mathsf{LT}\big(\tau_i^{\mathrm{mt}}\!\restriction\! A\big).$$

- ***Timing is forgotten (invariance under order-preserving retiming)***

$$\forall \langle t'^{i}_{k} \rangle \textit{ strictly increasing}: \quad \mathsf{LT}\big(\langle (a^i_0, t'^i_0), \ldots, (a^i_{n_i-1}, t'^i_{n_i-1}) \rangle\big) \;=\; \mathsf{LT}(\tau^{\mathrm{mt}}_i).$$

Proof. Construction. Define

$$\mathsf{LT}(\tau^{\mathrm{mt}}_i) := \langle (a^i_0),\, (a^i_1),\, \ldots,\, (a^i_{n_i-1}) \rangle.$$

All properties follow from erasing time stamps while preserving labels and their order. □

From metric time to synchronous time. The map ST aligns events to a global round clock by inserting "$-$" at rounds with no event; erasing "$-$" brings us back to logical time.

Example 6 (Motivation for ST: round-indexed view)**.**

$$\textit{Let } \tau^{\mathrm{mt}}_i = \langle (\textsf{\textit{badge\_tapped}}, 1), (\textsf{\textit{door\_open}}, 3) \rangle,$$
$$\textit{then } \mathsf{ST}(\tau^{\mathrm{mt}}_i) = \langle (-), (\textsf{\textit{badge\_tapped}}), (-), (\textsf{\textit{door\_open}}) \rangle.$$

With $|\mathsf{ST}(\tau^{\mathrm{mt}}_i)| = 4, \quad \mathsf{erase}_-(\mathsf{ST}(\tau^{\mathrm{mt}}_i)) = \langle (\textsf{\textit{badge\_tapped}}), (\textsf{\textit{door\_open}}) \rangle = \mathsf{LT}(\tau^{\mathrm{mt}}_i)$. *The set of active rounds is* $\{1, 3\} = \mathsf{time}(\tau^{\mathrm{mt}}_i)$.

Lemma 2 (Synchronous padding ST: round-indexed view)**.** *Let τ^{mt}_i be a metric-timed word, and assume one synchronous tick per metric unit. There exists a unique synchronous word $\tau^{\mathrm{st}}_i = \mathsf{ST}(\tau^{\mathrm{mt}}_i) \in (\Sigma_i \cup \{-\})^*$ such that, writing $T := \mathsf{last}(\tau^{\mathrm{mt}}_i)$ when $|\tau^{\mathrm{mt}}_i| > 0$:*

- ***Size / horizon relation***

$$|\mathsf{ST}(\tau^{\mathrm{mt}}_i)| = \begin{cases} T+1, & |\tau^{\mathrm{mt}}_i| > 0, \\ 0, & |\tau^{\mathrm{mt}}_i| = 0, \end{cases} \qquad \mathsf{pos}\big(\mathsf{ST}(\tau^{\mathrm{mt}}_i)\big) = \{0, 1, \ldots, T\} \;\textit{ if } |\tau^{\mathrm{mt}}_i| > 0.$$

- ***Exact time-of-occurrence***

$$\forall t \in \mathsf{pos}\big(\mathsf{ST}(\tau^{\mathrm{mt}}_i)\big): \quad \mathsf{ST}(\tau^{\mathrm{mt}}_i)[t] = \begin{cases} \mathsf{lab}(\tau^{\mathrm{mt}}_i, k), & \textit{if } t = \mathsf{ts}(\tau^{\mathrm{mt}}_i, k) \textit{ for some unique } k, \\ -, & \textit{otherwise.} \end{cases}$$

- ***The number of active rounds equals original event times***

$$\mathsf{act}\big(\mathsf{ST}(\tau^{\mathrm{mt}}_i)\big) \;=\; \{ t \mid \mathsf{ST}(\tau^{\mathrm{mt}}_i)[t] \neq - \} \;=\; \mathsf{time}(\tau^{\mathrm{mt}}_i).$$

- ***Preservation of the number of occurrences***

$$\forall a \in \Sigma_i: \quad \#_a\big(\mathsf{ST}(\tau^{\mathrm{mt}}_i)\big) \;=\; \#_a\big(\tau^{\mathrm{mt}}_i\big).$$

- ***Gaps realized as stutter length***

$$\forall k < n_i - 1: \quad \#\{\, t \mid t^i_k < t < t^i_{k+1},\ \mathsf{ST}(\tau^{\mathrm{mt}}_i)[t] = - \,\} \;=\; t^i_{k+1} - t^i_k - 1.$$

- ***Prefix-by-time commutation***

$$\forall T' \in \mathbb{N}: \qquad \mathsf{ST}(\tau^{\mathrm{mt}}_i){\restriction}[0..T'] \;=\; \mathsf{ST}\big(\tau^{\mathrm{mt}}_i{\restriction}\{t \le T'\}\big).$$

- ***Equivalence to logical time***

$$\mathsf{erase}_-\big(\mathsf{ST}(\tau^{\mathrm{mt}}_i)\big) \;=\; \mathsf{LT}(\tau^{\mathrm{mt}}_i).$$

Proof. Construction. If $|\tau^{\mathrm{mt}}_i| = 0$, set $\mathsf{ST}(\tau^{\mathrm{mt}}_i) := \langle - \rangle$. Otherwise let $T := \mathsf{last}(\tau^{\mathrm{mt}}_i)$ and define for every $t \in \{0, \dots, T\}$:

$$\mathsf{ST}(\tau^{\mathrm{mt}}_i)[t] \;=\; \begin{cases} a^i_k, & \text{if } t = t^i_k \text{ for the unique } k, \\ -, & \text{otherwise.} \end{cases}$$

Each property follows directly from this definition and the strict monotonicity of timestamps. □

On non-invertibility. Both LT and ST are *many-to-one*. In particular, LT is *not* invertible: for any fixed $\tau^{\mathrm{lt}}_i \in \Sigma^*_i$, there are infinitely many τ^{mt}_i (different timestamp choices) and τ^{st}_i (different stutter patterns/horizons) such that $\mathsf{LT}(\tau^{\mathrm{mt}}_i) = \tau^{\mathrm{lt}}_i$ and $\mathsf{erase}_-\ (\tau^{\mathrm{st}}_i) \;=\; \tau^{\mathrm{lt}}_i$. Hence, one cannot reconstruct a unique metric or synchronous trace from logical time alone. (Conversely, given $\mathsf{ST}(\tau^{\mathrm{mt}}_i)$ under the one-tick-per-unit assumption, the metric timestamps are readable as the indices of non-stutter symbols.)

Takeaway. LT and ST are the canonical forgetful maps from metric time to, respectively, order-only and round-synchronous views. They preserve exactly the event properties listed above and discard the rest. These morphisms will be used to state synchronization operators once and then instantiate them uniformly across models.

3.2 State of the Art Synchronization Operators

Design rationale. Different communities fix different *time assumptions*, which determine how local traces are aligned. Process calculi (CCS/CSP) adopt *no global clock* and reason over order only, yielding *pure interleaving* and optionally *shared action handshakes* [15, 12]. Synchronous languages assume a *single global round clock* (lockstep) [10]. Timed automata use *absolute timestamps* and synchronize at equal times on shared actions [2]. There are also *partial synchrony* notions via logical clocks (e.g. Lamport clocks) for message-passing systems [14]; we *do not* cover those here since we focus on action traces rather than message causality.

Below, we instantiate three operators consistent with our models: (i) *asynchrony* on logical-time words (τ^{lt}), (ii) *lockstep synchrony* on synchronous words (τ^{st}), and (iii) *synchronous handshake* on synchronous words with an explicit set A of shared actions.

3.2.1 Asynchrony Operator on Logical Time

The asynchronous operator models purely interleaved joint behavior, generating *all shuffles* that preserve each agent's local order. No global clock or rendezvous is assumed. To avoid accidental identification of simultaneous events, the global alphabet is taken as the *disjoint union* $\Sigma = \Sigma_1 \uplus \Sigma_2$, tagging each action with its agent of origin. This is the standard interleaving semantics of CCS and CSP [15, 12]. In true concurrency theory, the same construction underlies *Mazurkiewicz traces*, where equivalence classes of interleaving are taken modulo an independence relation I [6].

Definition 4 (Asynchronous interleaving on τ^{lt})**.** *Let $\tau_1^{\mathrm{lt}} \in \Sigma_1^*$ and $\tau_1^{\mathrm{lt}} \in \Sigma_2^*$ be logical-time words. The* asynchronous interleaving operator $\mid\mid\mid \; : \; \Sigma_1^* \times \Sigma_2^* \longrightarrow 2^{(\Sigma_1 \uplus \Sigma_2)^*}$ *is defined recursively by*

$$\tau_1^{\mathrm{lt}} \mid\mid\mid \tau_2^{\mathrm{lt}} = \begin{cases} \{\, \tau_2^{\mathrm{lt}} \}, & \tau_1^{\mathrm{lt}} = \langle - \rangle, \\ \{\, \tau_1^{\mathrm{lt}} \}, & \tau_2^{\mathrm{lt}} = \langle - \rangle, \\ \{\, a \cdot w \mid w \in (u \mid\mid\mid b \cdot \tau_2^{\mathrm{lt}}) \,\} \cup \{\, b \cdot w \mid w \in (a \cdot \tau_1^{\mathrm{lt}} \mid\mid\mid v) \,\}, & \tau_1^{\mathrm{lt}} = a \cdot u, \; \tau_2^{\mathrm{lt}} = b \cdot v. \end{cases}$$

Where $a \in \Sigma_1$, $b \in \Sigma_2$ and $u \in \Sigma_1^*$ and $v \in \Sigma_2^*$.

As a result, we have that every $w \in \mid\mid\mid(u, v)$ satisfies the projection property:

$$w{\restriction}\Sigma_1 = \tau_1^{\mathrm{lt}}, \qquad w{\restriction}\Sigma_2 = \tau_2^{\mathrm{lt}}.$$

Example 7 (Warehouse pick/scan, pure interleaving)**.** *Let $\Sigma_1 = \{\mathsf{pick}\}$, $\Sigma_2 = \{\mathsf{scan}\}$. Agent 1: $\tau_1^{\mathrm{lt}} = \langle (\mathsf{pick}), (\mathsf{pick}) \rangle$. Agent 2: $\tau_2^{\mathrm{lt}} = \langle (\mathsf{scan}) \rangle$. Then*

$$\tau_1^{\mathrm{lt}} \mid\mid\mid \tau_2^{\mathrm{lt}} = \Big\{ \langle (\mathsf{scan}), (\mathsf{pick}), (\mathsf{pick}) \rangle, \; \langle (\mathsf{pick}), (\mathsf{scan}), (\mathsf{pick}) \rangle, \; \langle (\mathsf{pick}), (\mathsf{pick}), (\mathsf{scan}) \rangle \Big\}.$$

All outputs project back: $w{\restriction}\Sigma_1 = \tau_1^{\mathrm{lt}}$, $w{\restriction}\Sigma_2 = \tau_2^{\mathrm{lt}}$.

Notes. If you insist on $\Sigma_1 \cap \Sigma_2 \neq \emptyset$ without tagging, projection equalities implicitly force "shared" symbols to coincide; to keep *pure* interleaving, use the tagged disjoint union $\uplus$.

3.2.2 Synchronous (Lockstep) Operator on Synchronous Time

The *lockstep* operator combines two synchronous traces by aligning them round by round under a shared global clock. If the two words have different horizons, the shorter one is right-padded with the stutter symbol "$-$" so that both align on the common index set $[0..T]$, where $T = \max(T_1, T_2)$. This operator reflects the synchronous hypothesis supported by languages such as Esterel and Lustre [10], in which all components react simultaneously at each logical tick.

Definition 5 (Lockstep zip on τ^{st})**.** *Let $\tau_1^{\mathrm{st}} = \langle (s_0), \ldots, (s_{T_1}) \rangle$ and $\tau_2^{\mathrm{st}} = \langle (r_0), \ldots, (r_{T_2}) \rangle$ be two synchronous words. Extend the shorter word with stutters "$-$" up to $T :=$* $\max(T_1, T_2)$*. The* lockstep zip *operator is defined by*

$$\tau_1^{\mathrm{st}} \parallel_{\mathrm{lock}} \tau_2^{\mathrm{st}} \;:=\; \langle\, (s_0, r_0), \; (s_1, r_1), \; \ldots, \; (s_T, r_T) \,\rangle.$$

Each position t of the result records the simultaneous round of both agents. Projections recover the original words:

$$(\tau_1^{\mathrm{st}} \parallel_{\mathrm{lock}} \tau_2^{\mathrm{st}}) \upharpoonright (\Sigma_1 \times \{-\} \cup \Sigma_1) = \tau_1^{\mathrm{st}}, \qquad (\tau_1^{\mathrm{st}} \parallel_{\mathrm{lock}} \tau_2^{\mathrm{st}}) \upharpoonright (\Sigma_2 \times \{-\} \cup \Sigma_2) = \tau_2^{\mathrm{st}}.$$

3.2.3 Synchronous Operator With Handshake Actions

In many systems, certain actions can only be executed *jointly* and must occur in *the same round* for both agents. Let $A \subseteq \Sigma_1 \cap \Sigma_2$ denote the set of *handshake actions*. The idea is that if one agent performs a handshake action $a \in A$ at some round, then the other agent must also perform a at that exact round. This principle, known from the semantics of Communicating Sequential Processes (CSP) [12], requires handshake actions to be treated as simultaneous, mutually synchronized events.

Definition 6 (Lockstep with handshakes on synchronous words)**.** *Let $A \subseteq \Sigma_1 \cap \Sigma_2$ be a nonempty set of* handshake actions*. The operator* lockstep with handshakes*, written*

$$\tau_1^{\mathrm{st}} \parallel_{\mathrm{hs}}^{A} \tau_2^{\mathrm{st}},$$

takes two synchronous words $\tau_1^{\mathrm{st}} = \langle (s_0), \ldots, (s_{T_1}) \rangle \in (\Sigma_1 \cup \{-\})^*$ and $\tau_2^{\mathrm{st}} = \langle (r_0), \ldots, (r_{T_2}) \rangle \in (\Sigma_2 \cup \{-\})^*$, pads them to the common horizon $T = \max(T_1, T_2)$, and returns

$$\tau_1^{\mathrm{st}} \parallel_{\mathrm{hs}}^{A} \tau_2^{\mathrm{st}} \;=\; \langle (s_0, r_0), (s_1, r_1), \ldots, (s_T, r_T) \rangle.$$

The operator is defined *if and only if the following handshake constraint holds:*

$$\forall t \in \{0, \ldots, T\},\ \forall a \in A : \quad \big(s_t = a \,\vee\, r_t = a\big) \;\Rightarrow\; (s_t = r_t = a).$$

Otherwise, if there exists some t and $a \in A$ such that exactly one of s_t, r_t equals a, the operator is undefined*. In other words, every handshake action must be executed simultaneously by both agents, whereas private actions and stuttering symbols "$-$" may occur independently.*

Definition 7 (Collapsed lockstep with handshakes)**.** *For the same setting as above, define $\Sigma := \Sigma_1^{(1)} \uplus \Sigma_2^{(2)} \uplus A$. The* collapsed lockstep with handshakes *is obtained by applying the morphism $\mathsf{coll}_A : \big((\Sigma_1 \cup \{-\}) \times (\Sigma_2 \cup \{-\})\big)^* \to (\Sigma \cup \{-\})^*$, defined round-by-round as*

$$\mathsf{coll}_A(s_t, r_t) = \begin{cases} a, & \text{if } s_t = r_t = a \in A, \\ s_t^{(1)}, & \text{if } s_t \in \Sigma_1 \setminus A,\ r_t = -, \\ r_t^{(2)}, & \text{if } r_t \in \Sigma_2 \setminus A,\ s_t = -, \\ (s_t^{(1)}, r_t^{(2)}), & \text{if } s_t \in \Sigma_1 \setminus A,\ r_t \in \Sigma_2 \setminus A, \\ -, & \text{if } s_t = r_t = -. \end{cases}$$

Thus, joint actions from A collapse to a single shared letter, while private actions remain tagged by their originating agent.

Example 8 (Handover as handshake, failures and successes)**.** *Let $\Sigma_1 = \{\mathit{pick}, \mathit{handover}\}$, $\Sigma_2 = \{\mathit{scan}, \mathit{handover}\}$, $A = \{\mathit{handover}\}$.*

- Success (aligned handshake).
  $\tau_1^{\mathrm{st}} = \langle (\mathit{pick}), -, \mathit{handover}\rangle, \quad \tau_2^{\mathrm{st}} = \langle -, \mathit{scan}, \mathit{handover}\rangle.$
  Then $\tau_1^{\mathrm{st}} \,\|_{\mathrm{hs}}^{A}\, \tau_2^{\mathrm{st}} = \langle (\mathit{pick}, -),\ (-, \mathit{scan}),\ (\mathit{handover}, \mathit{handover})\rangle$*, and*
  $\mathsf{coll}_A(\tau_1^{\mathrm{st}} \,\|_{\mathrm{hs}}^{A}\, \tau_2^{\mathrm{st}}) = \langle (\mathit{pick}^{(1)}),\ (\mathit{scan}^{(2)}),\ (\mathit{handover})\rangle.$

- Failure (mismatched handshake).
  $\tau_1^{\mathrm{st}} = \langle -, \mathit{handover}, -\rangle, \quad \tau_2^{\mathrm{st}} = \langle \mathit{handover}, -, -\rangle.$
  At round 0 *agent 2 emits* handover *while agent 1 does not; the constraint is violated, so* $\tau_1^{\mathrm{st}} \,\|_{\mathrm{hs}}^{A}\, \tau_2^{\mathrm{st}}$ *is* undefined.

- Private simultaneous actions (non-handshake).
  $\tau_1^{\mathrm{st}} = \langle -, \mathit{pick}, -\rangle, \quad \tau_2^{\mathrm{st}} = \langle -, \mathit{scan}, -\rangle.$
  Since neither label is in A, the pair $(\mathit{pick}, \mathit{scan})$ *is allowed at the same round:* $\tau_1^{\mathrm{st}} \,\|_{\mathrm{hs}}^{A}\, \tau_2^{\mathrm{st}} = \langle (-,-),\ (\mathit{pick}, \mathit{scan}),\ (-,-)\rangle$*. Under* coll_A*, this becomes two tagged letters in that round.*

In summary, we analyzed three synchronization operators, each situated in the semantic model where it is most naturally defined. The asynchronous operator $|||$ captures all globally feasible interleavings that respect local order, without assuming any global notion of time or coordination. By contrast, the lockstep operator $\|_{\mathrm{lock}}$ assumes a shared global clock: it aligns traces round by round and makes stuttering explicit. Finally, $\|_{\mathrm{hs}}^{A}$ strengthens the lockstep view by enforcing handshake constraints on the designated set A of joint actions. When both agents perform the same handshake action simultaneously, the pair can be collapsed into a single shared symbol through the morphism coll_A.

These operators represent the state of the art in computer science for defining the semantics of modal, temporal, and strategic operators over multi-agent executions under different synchrony assumptions. The following section introduces a new model that is better suited for reasoning about normative systems as collaborative specifications. Some aspects of the synchronous operator with handshake actions are reused.

3.3 First Contribution: Attempted/Successful Abstraction

3.3.1 What Do We Want to Model?

The aim is to capture two complementary facets of collaboration between agents that standard system-centric models overlook, as illustrated in Example 1:

- **Negative performance (non-interference).** One agent must *refrain* from actions that would prevent the other from achieving a legitimate objective. In a tenant–landlord scenario, the landlord should not perform blocking actions such as $\mathsf{cut\_power}$, $\mathsf{change\_lock}$, or $\mathsf{enter\_flat}$, which would hinder the tenant's ability to occupy the flat.

- **Active participation (positive performance).** An agent must *contribute* and perform actions that make the objective of the counterpart achievable within the agreed period. For example, the landlord should provide/confirm $\mathsf{account\_info}$, ack_pay, and grant_occ upon settlement; symmetrically, the tenant should pay_rent and not return_payment.

These performance aspects are common in normative / contract settings but are underexplored by classical synchronization operators (interleaving, lockstep, handshake), which specify *how* events align rather than whether agents *refrain* from harmful actions or *contribute* enabling ones.

3.3.2 From Timed Words Over Actions to Periodic Synchronous Words Over Sets of Actions

Example 9 (Timed words only (agents Tenant(1) and Landlord(2)))**.**

$$\Sigma_1 = \{\,\mathit{pay\_rent},\ \mathit{pay\_rent\_f},\ \mathit{start\_occ}\,\mathit{stop\_occupy}\},$$
$$\Sigma_2 = \{\,\mathit{ack\_pay},\ \mathit{grant\_occ},\ \mathit{ref\_pay},\ \mathit{change\_lock}\,\}.$$

Scenario (Four-Month Interaction). Over four months (days $t \in \{0,\dots,120\}$), the landlord grants occupancy on day 1 (grant_occ). The tenant does not move in during the first month, but pays rent on day 5 (pay_rent), acknowledged on day 6 (ack_pay). In the second month, the tenant begins occupying the flat (start_occ) on day 35 after the start of the contract; the landlord posts an administrative acknowledgement on day 37 (ack_pay) without a new rent event that month. In month 3, The tenant continues to occupy, but this month the tenant does not pay; the landlord takes no action. In the fourth month, the tenant pays a late fee on day 95 (pay_rent_f); the landlord refuses that payment the same day (ref_pay) and changes the lock on day 96 (change_lock).

Metric-timed words.

$$\tau_1^{\mathrm{mt}} = \langle(\mathit{pay\_rent},5),\ (\mathit{start\_occ},35),\ (\mathit{pay\_rent\_f},95)\rangle,$$
$$\tau_2^{\mathrm{mt}} = \langle(\mathit{grant\_occ},1),\ (\mathit{ack\_pay},6),\ (\mathit{ack\_pay},37),$$
$$(\mathit{ref\_pay},95),\ (\mathit{change\_lock},96)\rangle.$$

From days to months: Why periodize? Since the contract regulates obligations and permissions *per month* rather than per day, we periodize the timeline into month windows $I_0, I_1, I_2, I_3, \dots$ and aggregate the actions of each agent within the same window into a *set*. This yields a synchronous round-based abstraction aligned with the contract calendar.

Definition 8 (Periodic synchronized set trace)**.** *A* periodic synchronized set trace *for agent i over alphabet Σ_i is a (finite or infinite) sequence of per-period action sets*

$$\pi_i \;=\; \langle A_0^{(i)},\, A_1^{(i)},\, A_2^{(i)},\, \dots\rangle \;\in\; (2^{\Sigma_i})^* \;\; (\mathit{or}\ (2^{\Sigma_i})^{\omega}).$$

Each $A_k^{(i)} \subseteq \Sigma_i$ represents the set of actions attributed to agent i during the the period k (empties allowed).

Lemma 3 (Conversion from metric time to periodic synchronized trace)**.** *Let* $\tau_i^{\mathrm{mt}} \in (\Sigma_i \times \mathbb{N})^*$ *be a (finite) timed word for agent i, and let* $(I_k)_{k\in\mathbb{N}}$ *be a partition of* $\mathbb{N}$ *into disjoint, totally ordered* period windows *(e.g., ongoing months) such that*

$$k < \ell,\ t \in I_k,\ s \in I_\ell \ \Rightarrow\ t < s.$$

Then there exists a periodic synchronized timed trace $\pi_i = \langle A_0^{(i)}, \ldots, A_K^{(i)} \rangle \in (2^{\Sigma_i})^*$ *defined by*

$$A_k^{(i)} \ :=\ \{ a \in \Sigma_i \mid \exists t \in I_k : \ (a,t) \in \tau_i^{\mathrm{mt}} \}.$$

Moreover, the aggregation preserves inter-period order*:*

$$\forall k\ \forall a \in A_k^{(i)}\ \forall b \in A_{k+1}^{(i)}\ \ \exists t_a \in I_k,\ t_b \in I_{k+1} : \ (a,t_a) \in \tau_i^{\mathrm{mt}},\ (b,t_b) \in \tau_i^{\mathrm{mt}},\ \textit{and}\ t_a < t_b.$$

Proof. By definition of Agg_I. Given $a \in A_k^{(i)}$ and $b \in A_\ell^{(i)}$ with $k < \ell$, choose witnesses $t_a \in I_k$ and $t_b \in I_\ell$; the window order yields $t_a < t_b$. □

Nevertheless, aggregation has a cost, as it loses intra-period order, action multiplicity, and exact timestamps. Therefore, selecting the appropriate period normalization is crucial.

Example 10 (Transforming timed words into periodic set traces, continued from Example. 9)**.** *By fixing the period to 30, we can decompose the 120 days into 4 horizons:*

$$I_0 = [0,30], \qquad I_1 = [31,60], \qquad I_2 = [61,90], \qquad I_3 = [91,120].$$

The two timed traces τ_1^{mt} *and* τ_2^{mt} *could be transformed into:*

$$\pi_1 = \langle \underbrace{\{\mathit{pay\_rent}\}}_{I_0}, \underbrace{\{\mathit{start\_occ}\}}_{I_1}, \underbrace{\emptyset}_{I_2}, \underbrace{\{\mathit{pay\_rent\_f}\}}_{I_3} \rangle,$$

$$\pi_2 = \langle \underbrace{\{\mathit{grant\_occ},\ \mathit{ack\_pay}\}}_{I_0}, \underbrace{\{\mathit{ack\_pay}\}}_{I_1}, \underbrace{\emptyset}_{I_2}, \underbrace{\{\mathit{ref\_pay\_f},\ \mathit{change\_lock}\}}_{I_3} \rangle.$$

By Lemma 3, for agent 1 the element $\mathit{start\_occ} \in A_1^{(1)}$ *precedes* $\mathit{pay\_rent\_f} \in A_3^{(1)}$ *(witness times* $35 < 95$*), and similarly for agent 2.*

3.3.3 The Abstraction of Attempted and Declined Interactions Over Synchronized Alphabets

The second aspect of the abstraction is the notion of interaction alphabets. The action alphabets of both agents are merged into a single alphabet, and the trace encodes that when a is an event, it indicates that an agent has either instantiated it or accepted to collaborate on it. Conversely, the absence of an action from the event signifies that the agent either did not perform it or actively performed another action to prevent its success. This abstraction is demonstrated using the running example, showing that it maintains the same meaning while reducing the number of action types. An operator is then introduced to compute the successful interaction at each period of the synchronized trace. This transformation cannot be automated unless the set of enabling and interfering actions is explicitly stated in the normative specification; otherwise, the digitization engineer must define them manually, as is done here.

We consider three collaborative objectives that are identified in the Example.1: $\mathsf{PAY\_R}$ (rent payment), $\mathsf{PAY\_F}$ (penalty/late-fee payment), and OCC (tenant's occupancy). We write $\Sigma_C = \{\mathsf{PAY\_R}, \mathsf{PAY\_F}, \mathsf{OCC}\}$.

In the next step, we need to define how the actions of the agents, namely Σ_1 and Σ_2, relate to collaborative action over Σ_C, more specifically, whether they are enabler or interference actions. We do not take the union $\Sigma_1 \cup \Sigma_2$ as a synchronization operator discussed in the Subsection 3.2. Instead, we define a many-to-one *abstraction* from concrete per-agent actions to the collaboration alphabet Σ_C. On the tenant side, $\mathsf{pay\_rent}$ and $\mathsf{pay\_rent\_f}$ are enablers because they instantiate the tenant's contribution toward $\mathsf{PAY\_R}$ and $\mathsf{PAY\_F}$; $\mathsf{start\_occ}$ is an enabler for OCC because it is the tenant's side of taking possession. A chargeback $\mathsf{return\_payment}$ and $\mathsf{return\_payment\_f}$ is blocking: it nullifies the very transfer that $\mathsf{PAY\_R}$ and *PAY_F* rely on. On the landlord side, $\mathsf{ack\_pay}$ enables $\mathsf{PAY\_R}$, while $\mathsf{grant\_occ}$ enables OCC by authorizing access. In contrast, $\mathsf{ref\_pay}$ blocks $\mathsf{PAY\_R}$ even if the tenant initiated payment, and $\mathsf{change\_lock}$, $\mathsf{cut\_power}$, or $\mathsf{enter\_flat}$ block OCC by making continued possession impracticable or unlawful. This enabler/blocker partition is precisely what our "suggested/successful" abstraction needs: success in a period occurs when both sides propose the required enablers and neither side performs a blocker.

Distinguishing enabling from Blocking actions For each agent $i \in \{1,2\}$ we partition the alphabet into Σ_i^A (actions that constitute *active participation*, i.e., enablers) and Σ_i^I (actions that constitute *negative performance*, i.e., blockers), with $\Sigma_i = \Sigma_i^A \uplus \Sigma_i^I$ (disjoint union). In our running example:

$$\Sigma_1^A = \{\,\mathsf{pay\_rent},\ \mathsf{pay\_rent\_f},\ \mathsf{start\_occ}\,\},$$
$$\Sigma_1^I = \{\,\mathsf{refuse\_inspection}\,\}.$$

$$\Sigma_2^A = \{\,\mathsf{ack\_pay},\ \mathsf{grant\_occ}\,\},$$
$$\Sigma_2^I = \{\,\mathsf{ref\_pay},\ \mathsf{change\_lock},\ \mathsf{ref\_pay\_f},\ \mathsf{enter\_flat}\,\}.$$

$$\Sigma_1 = \Sigma_1^A \uplus \Sigma_1^I, \qquad \Sigma_2 = \Sigma_2^A \uplus \Sigma_2^I, \qquad \Sigma = \Sigma_1 \cup \Sigma_2.$$

For brevity, below we write the *enabler* and *blocker* sets as

$$E_i := \Sigma_i^A \quad \text{and} \quad B_i := \Sigma_i^I \qquad (i \in \{1,2\}).$$

Presence of $a \in E_i$ in period k signals that agent i took a *positively contributing* action; presence of $b \in B_i$ signals a *defeating* (interfering) action. The absence of a symbol indicates it was not suggested/endorsed during that period.

Transformation sketch After defining this relation, we can transform any periodic synchronous word π_1 over Σ_1 and π_2 over Σ_2 to a corresponding word over Σ_C, written π_i^C:

- If an action a is on event A from π_i and that action is in E_i, then this action is transformed to its equivalent collaborative action and inserted on the resulting word π_i^C.
- If an action a is on event A from π_i and that action is in B_i, then this action is not transformed and not inserted on the resulting word π_i^C.

And additionally, we do not add any collaborative action not present in an event on the equivalent event in π_i^C unless it is a continuous collaboration with implicit collaboration, as start_occ signals the start of occupying the flat, so it is kept inserted in the following events in the timed word of the tenant and landlord as long as tenant does not leave nor the landlord blocks actively the occupation.

Example 11 (Transforming the periodic synchronized words over Σ_1, Σ_2 to periodic synchronized words over Σ_C)**.** *Using the traces from Example 10,*

$$\pi_1 = \langle \underbrace{\{\mathit{pay\_rent}\}}_{I_0}, \underbrace{\{\mathit{start\_occ}, \mathit{pay\_rent}\}}_{I_1}, \underbrace{\emptyset}_{I_2}, \underbrace{\{\mathit{pay\_rent\_f}\}}_{I_3}\rangle,$$

$$\pi_2 = \langle \underbrace{\{\mathit{grant\_occ},\ \mathit{ack\_pay}\}}_{I_0}, \underbrace{\{\mathit{ack\_pay}\}}_{I_1}, \underbrace{\emptyset}_{I_2}, \underbrace{\{\mathit{ref\_pay\_f},\ \mathit{change\_lock}\}}_{I_3}\rangle.$$

With $E_1(\mathsf{PAY\_R}) = \{\mathit{pay\_rent}\}$,$E_1(\mathsf{PAY\_F}) = \{\mathit{pay\_rent\_f}\}$ $E_2(\mathsf{PAY\_R}) = \{\mathit{ack\_pay}\}$, $B_2(\mathsf{PAY\_R}) = \{\mathit{ref\_pay}\}$, $E_1(\mathsf{OCC}) = \{\mathit{start\_occ}\}$, $E_2(\mathsf{OCC}) = \{\mathit{grant\_occ}\}$, $B_2(\mathsf{OCC}) = \{\mathit{change\_lock}\}$*, Consequently, the collaboration alphabet is:*

$$\Sigma_C = \{\mathsf{PAY\_R},\ \mathsf{PAY\_F},\ \mathsf{OCC}\}.$$

The corresponding collaboration-trace abstractions of π_1 *and* π_2 *are* π_1^C *and* π_2^C*:*

$$\pi_1^C = \langle \underbrace{\{\mathsf{PAY\_R}^{(1)}\}}_{I_0}, \underbrace{\{\mathsf{OCC}^{(1)},\ \mathsf{PAY\_R}^{(1)}\}}_{I_1}, \underbrace{\{\mathsf{OCC}^{(1)}\}}_{I_2}, \underbrace{\{\mathsf{PAY\_F}^{(1)}\}}_{I_3}\rangle,$$

$$\pi_2^C = \langle \underbrace{\{\mathsf{OCC}^{(2)},\ \mathsf{PAY\_R}^{(2)}\}}_{I_0}, \underbrace{\{\mathsf{OCC}^{(2)},\ \mathsf{PAY\_R}^{(2)}\}}_{I_1}, \underbrace{\{\mathsf{OCC}^{(2)}\}}_{I_2}, \underbrace{\emptyset}_{I_3}\rangle.$$

Reading: in month I_0*, agent 1 positively contributes to* PAY_R*, and agent 2 allows* OCC *and accepts* PAY_R*. In* I_1*, agent 1 pays the rent and occupies the flat and agent 2 contributes to* PAY_R *and allows the occupation* OCC*. In* I_2*, agent 1 keeps* OCC *but does not pay, and the landlord keeps allowing* OCC*. In* I_3*, the landlord blocks both* PAY_R *and denies* OCC*, although the tenant wants to keep occupying the flat and pays.*

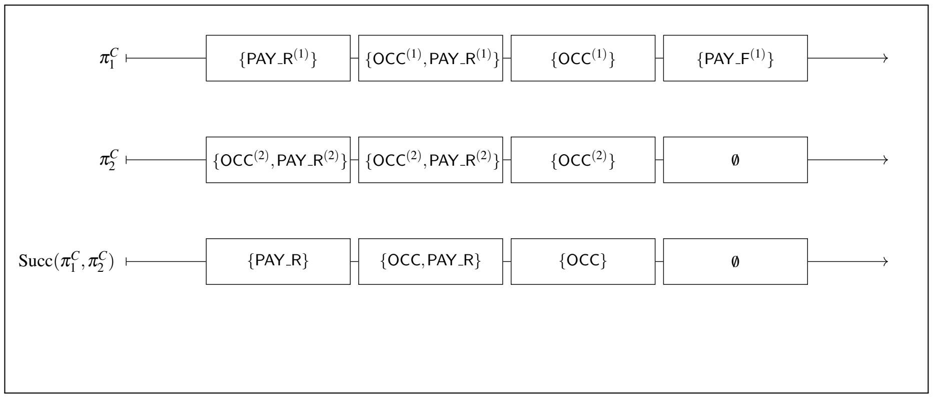
3.3.4 Successful Action Computation From Two Agents' Interaction

Definition 9 (Successful collaboration operator)**.** *Let* Σ *be an alphabet and let* $\Sigma^{(1)} := \{a^{(1)} \mid a \in \Sigma\}$ *and* $\Sigma^{(2)} := \{a^{(2)} \mid a \in \Sigma\}$ *be disjoint tagged copies. The* successful collaboration operator, *written* Succ *of two periodic synchronous trace* π_1 *over* $\Sigma^{(1)}$ *and* π_2 *over* $\Sigma^{(2)}$ *on the same maximum horizon* T *is defined eventwise by:*

$$\forall k \in \{0,\dots,T\}, \mathrm{Succ}(\pi_1,\pi_2)[k] := \mathsf{unlab}\big(\pi_1[k]\big) \cap \mathsf{unlab}\big(\pi_2[k]\big).$$

where unlab *is the function removing the agent identifier tag from an event.*

Example 12 (Successful collaboration)**.** *From the two transformed traces* π_1^C *and* π_2^C *from the Example.11 We illustrate* $\mathrm{Succ}(\pi_1^C,\pi_2^C)$ *in Figure. 1*



π_1^C
$\{\mathsf{PAY\_R}^{(1)}\}$
$\{\mathsf{OCC}^{(1)},\mathsf{PAY\_R}^{(1)}\}$
$\{\mathsf{OCC}^{(1)}\}$
$\{\mathsf{PAY\_F}^{(1)}\}$
π_2^C
$\{\mathsf{OCC}^{(2)},\mathsf{PAY\_R}^{(2)}\}$
$\{\mathsf{OCC}^{(2)},\mathsf{PAY\_R}^{(2)}\}$
$\{\mathsf{OCC}^{(2)}\}$
$\emptyset$
$\mathrm{Succ}(\pi_1^C,\pi_2^C)$
$\{\mathsf{PAY\_R}\}$
$\{\mathsf{OCC},\mathsf{PAY\_R}\}$
$\{\mathsf{OCC}\}$
$\emptyset$

Figure 1: Example: successful collaboration computation example

Basic properties. Since Succ is pointwise set intersection after tag erasure, it inherits three immediate facts: *(i) Commutative and idempotent*: $\mathrm{Succ}(\pi_1,\pi_2) = \mathrm{Succ}(\pi_2,\pi_1)$ and $\mathrm{Succ}(\pi,\pi) = \mathsf{unlab}(\pi)$. *(ii) Monotone (pointwise* $\subseteq$*)*: writing $\pi \le \pi'$ to mean $\forall k$: $\mathsf{unlab}(\pi[k]) \subseteq \mathsf{unlab}(\pi'[k])$, if $\pi_1 \le \pi_1'$ and $\pi_2 \le \pi_2'$ then $\mathrm{Succ}(\pi_1,\pi_2) \le \mathrm{Succ}(\pi_1',\pi_2')$. *(iii) Absorbing empty trace*: if $\forall k,\ \pi_2[k] = \emptyset$, then $\forall k,\ \mathrm{Succ}(\pi_1,\pi_2)[k] = \emptyset$.

Remark 1. *Relation to* $\|_{\mathrm{hs}}^{A}$*. If we declare every collaborative objective to be a handshake,* $A = \Sigma_C$*, then the lockstep-with-handshakes product enforces that* PAY_R*,* PAY_F*,* OCC *can only appear at a period when both agents present the same letter. Concretely, if we view each period's set* $A_k^{(i)}$ *as the multiset of letters occurring "at that round" and apply* $\|_{\mathrm{hs}}^{A}$ *round wise, then after collapsing paired handshakes to a single shared symbol (via* coll_A*) and unlabeling, We obtain exactly the success:*

$$\mathrm{Succ}(\pi_1,\pi_2) \; = \; \mathsf{unlab}\big(\mathsf{coll}_{\Sigma_C}\big(\pi_1 \|_{\mathrm{hs}}^{\Sigma_C} \pi_2\big)\big),$$

i.e., Succ *is the set-theoretic intersection semantics of lockstep-plus-handshakes on the collaboration alphabet.*

3.4 Second Contribution: Modeling Agent Interaction Strategies

Instead of reconstructing compliance post hoc from past events, the approach models how agents intend to behave in the future. Agents may wish to run their intended strategies against the contract to determine compliance, prevent future violations, and identify how other parties may potentially obstruct or exploit the contract.

Because collaborative success depends on both parties, each agent devises a strategy conditioned on the observed behavior of the other. For example, the landlord may plan differently depending on whether the tenant pays, while the tenant may stop paying if the landlord prevents continued occupancy or fails to repair reported damages.

3.4.1 Models for Interaction Strategies

We capture interaction strategies with *input/output* models that operate at the period granularity. At each period k, each agent i observes the other agent's period-k output and updates its internal state to produce its own period-$(k+1)$ output. We use *Moore machines* so that an agent's output at period k depends only on its current state (perfect monitoring with one-period delay). A synchronous feedback composition couples the two machines.

Definition 10 (Deterministic Moore machine)**.** *A deterministic Moore machine for agent i is a 6-tuple*

$$M_i \;=\; (S_i, s_i^0,\; \Sigma_I^i,\; \Sigma_O^i,\; \delta_i,\; \lambda_i),$$

where:

- S_i *is a finite set of states with initial state* $s_i^0 \in S_i$*;*
- Σ_I *is the input alphabet (the other agent's* untagged *collaborative letters);*
- Σ_O^i *is the set of output alphabet;*
- $\delta_i : S_i \times 2^{\Sigma_I} \to S_i$ *is a deterministic transition function;*
- $\lambda_i : S_i \to 2^{\Sigma_O}$ *is the output function.*

Given an input stream $X = (X_0, X_1, \dots)$ *with* $X_k \subseteq \Sigma_I$*, the induced run is* $s_i^0, s_i^1, \dots$ *with* $s_i^{k+1} = \delta_i(s_i^k, X_k)$ *and outputs* $Y_k = \lambda_i(s_i^k)$*.*

Definition 11 (Run and output of a deterministic Moore machine)**.** *Let*

$$M_i \;=\; (S_i, s_i^0,\; \Sigma_I^i,\; \Sigma_O^i,\; \delta_i,\; \lambda_i)$$

be a deterministic Moore machine for agent i as in Definition 10. An input stream *for* M_i *is a finite or infinite sequence* $X = (X_0, X_1, \dots)$ *with* $X_k \subseteq \Sigma_I^i$ *for all positions k.*

Run. *The* run of M_i on X *is the unique sequence of states*

$$\rho_i(X) \;=\; (s_i^0, s_i^1, s_i^2, \dots)$$

inductively defined by

$$s_i^0 := s_i^0, \qquad s_i^{k+1} := \delta_i(s_i^k, X_k) \quad \textit{for all } k \geq 0.$$

For finite input X of length $n+1$ *we write* $|X| = n+1$ *and* $\rho_i(X) = (s_i^0, \dots, s_i^{n+1})$*.*

Extended transition function. *For later use, we define the extended transition function*

$$\delta_i^* : S_i \times (2^{\Sigma_I^i})^* \to S_i$$

by

$$\delta_i^*(s, \varepsilon) := s, \qquad \delta_i^*(s, X_0 \cdot X') := \delta_i^*\big(\delta_i(s, X_0), X'\big),$$

for $X_0 \in 2^{\Sigma_I^i}$ *and* $X' \in (2^{\Sigma_I^i})^*$. *For a finite input word* X*, we write*

$$\delta_i(s_i^0, X) \ := \ \delta_i^*(s_i^0, X),$$

so that the last state of the run on X *is* $\delta_i(s_i^0, X)$.

Output word and terminal output. *The* output word *induced by* M_i *on* X *is*

$$\lambda_i^{\omega}(X) \ := \ (Y_0, Y_1, \dots) \quad \textit{with} \quad Y_k := \lambda_i(s_i^k).$$

For a finite input word X *the* terminal output *of* M_i *on* X *is*

$$\lambda_i\big(\delta_i(s_i^0, X)\big),$$

which is the output associated with the last state of the run on X.

3.4.2 Interactive Strategy Computation

We present the property of two Moore machines that can feed each other and progress together.

Definition 12 (Complementary Moore machines)**.**

$$\textit{Let } M_i \ = \ (S_i, s_i^0,\ \Sigma_I^i,\ \Sigma_O^i,\ \delta_i,\ \lambda_i) \textit{ and } M_j \ = \ (S_j, s_j^0,\ \Sigma_I^j,\ \Sigma_O^j,\ \delta_j,\ \lambda_j)$$

be two deterministic Moore machines, we say that M_i and M_j are complementary *if and only if:*

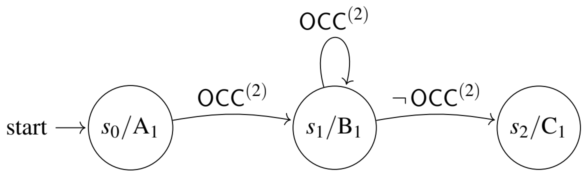
$$\Sigma_I^i = \Sigma_O^j \textit{ and } \Sigma_O^i = \Sigma_I^j.$$

In the following, we introduce an example of how to use Moore machines to capture two strategies that the landlord and the tenant should consider in the motivating example.

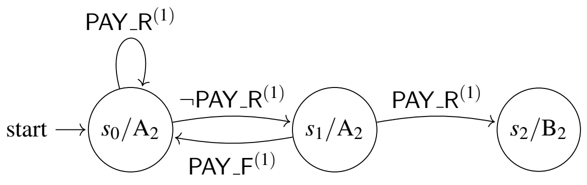
Example 13 (Interaction strategies and their Moore encodings)**.** *Consider the two first informal strategies* $\mathfrak{S}_1^1$ *and* $\mathfrak{S}_2^1$ *from respectively the tenant(1) and the landlord(2):*

Tenant $\mathfrak{S}_1^1$**.** *"I pay in the first month; from the second month on, I occupy and keep paying as long as the landlord does not stop me. If the landlord stops my occupancy, I stop paying."*

Landlord $\mathfrak{S}_2^1$**.** *"I enable occupancy from the first month and accept payment; if the tenant fails to pay for* two consecutive *months, I stop enabling occupancy." We encode both of those strategies using:* $\Sigma_C^{(1)} := \{a^{(1)} \mid a \in \Sigma_C\}$ *and* $\Sigma_C^{(2)} := \{a^{(2)} \mid a \in \Sigma_C\}$ *be the tagged disjoint copies. Both machines use* $S = \{s_0, s_1, s_2\}$ *with initial state* s_0*. Transitions are guarded by the current letters of the agent* other *(seen as a set).*



(a) M_1^1 (tenant).
$\mathrm{A}_1 = \{\mathsf{PAY\_R}^{(1)}\}$, $\mathrm{B}_1 = \{\mathsf{OCC}^{(1)}, \mathsf{PAY\_R}^{(1)}\}$, $\mathrm{C}_1 = \emptyset$.



(b) M_2^1 (landlord). Outputs: $\mathrm{A}_2 = \{\mathsf{OCC}^{(2)}, \mathsf{PAY\_R}^{(2)}\}$, $\mathrm{B}_2 = \emptyset$.

Figure 2: Moore machines for the tenant and landlord representing their strategies over tagged Σ_C. In the transition, the $\neg\mathsf{PAY\_R}^i$ in the Moore machine M_j is a shorthand for any input from M_i not containing $\mathsf{PAY\_R}^i$, similarly $\mathsf{PAY\_R}^i$ is a shorthand for any input from M_i containing $\mathsf{PAY\_R}^i$

Formalization (matching Fig. 2).

$$M_1^1 = (S, s_0,\ \Sigma_I^{(1)}, \Sigma_O^{(1)},\ \delta_1, \lambda_1), \quad \Sigma_I^{(1)} = 2^{\Sigma_C^{(2)}}, \quad \Sigma_O^{(1)} = 2^{\Sigma_C^{(1)}},$$
$$M_2^1 = (S, s_0,\ \Sigma_I^{(2)}, \Sigma_O^{(2)},\ \delta_2, \lambda_2), \quad \Sigma_I^{(2)} = 2^{\Sigma_C^{(1)}}, \quad \Sigma_O^{(2)} = 2^{\Sigma_C^{(2)}}.$$

For $X \subseteq \Sigma_C^{(2)}$, $Y \subseteq \Sigma_C^{(1)}$:

$$\delta_1(s_0, X) = \begin{cases} s_1 & \mathsf{OCC}^{(2)} \in X \\ s_0 & \textit{otherwise} \end{cases}, \quad \delta_1(s_1, X) = \begin{cases} s_1 & \mathsf{OCC}^{(2)} \in X \\ s_2 & \textit{otherwise} \end{cases}, \quad \delta_1(s_2, X) = s_2,$$

$$\delta_2(s_0, Y) = \begin{cases} s_0 & \mathsf{PAY\_R}^{(1)} \in Y \\ s_1 & \textit{otherwise} \end{cases}, \quad \delta_2(s_1, Y) = \begin{cases} s_0 & \mathsf{PAY\_F}^{(1)} \in Y \\ s_2 & \mathsf{PAY\_R}^{(1)} \in Y \\ s_1 & \textit{otherwise} \end{cases}, \quad \delta_2(s_2, Y) = s_2.$$

Notice that two Moore machines are complementary.

Now we move to the step where both strategies are fixed and transformed to compute their outcome. To do so, we introduce the product of two complementary Moore machines.

Definition 13 (Product of Complementary Determinstic Moore Machines)**.**

Let $M_i = (S_i, s_i^0,\ \Sigma_I^j,\ \Sigma_O^i,\ \delta_i,\ \lambda_i)$ *and* $M_j = (S_j, s_j^0,\ \Sigma_I^i,\ \Sigma_O^j,\ \delta_j,\ \lambda_j)$

be two deterministic complementary Moore machines. The product *of M_i and M_j is the automaton*

$$M_i \otimes M_j \;=\; (\Sigma,\, Q,\, q_0,\, \delta,\, F\,)$$

where

- $\Sigma = 2^{\Sigma_O^i \cup \Sigma_O^j}$ *is the* joint alphabet*,*
- $Q = S_i \times S_j$ *is the* state space*,*
- $q_0 = (s_i^0, s_j^0)$ *is the* initial state*,*
- $F = \emptyset$ *,*
- $\delta \subseteq Q \times \Sigma \times Q$ *is the* transition relation*, defined by*

  $$\big((s_i, s_j), A, (s_i', s_j')\big) \in \delta$$

  if and only if

  $$A = \lambda_i(s_i) \cup \lambda_j(s_j), \quad s_i' = \delta_i(s_i, \lambda_j(s_j)), \quad s_j' = \delta_j(s_j, \lambda_i(s_i)).$$

The language $L(M_i \otimes M_j) \subseteq \Sigma^\omega$ *consists of all infinite words*

$$\langle A_0, A_1, A_2 \ldots \rangle$$

such that there exists a run

$$(s_i^0, s_j^0) \xrightarrow{A_0} (s_{i,1}, s_{j,1}) \xrightarrow{A_1} (s_{i,2}, s_{j,2}) \xrightarrow{A_2} \ldots$$

with $A_k = \lambda_i(s_{i,k}) \cup \lambda_j(s_{j,k})$ for all $k \geq 0$.

Lemma 4 (Unique run and word of product machines)**.** *Let*

$$M_i = (S_i, s_i^0,\, \Sigma_I^j,\, \Sigma_O^i,\, \delta_i,\, \lambda_i) \quad \textit{and} \quad M_j = (S_j, s_j^0,\, \Sigma_I^i,\, \Sigma_O^j,\, \delta_j,\, \lambda_j)$$

be two complementary deterministic Moore machines. Then their product $M_i \otimes M_j$ admits a unique run

$$(s_i^0, s_j^0)\,(s_{i,1}, s_{j,1})\,(s_{i,2}, s_{j,2}) \ldots$$

and this run induces a unique word

$$\pi \;=\; \langle A_0, A_1, A_2, \ldots \rangle \;\in\; \big(2^{\Sigma_O^i \cup \Sigma_O^j}\big)^\omega,$$

where

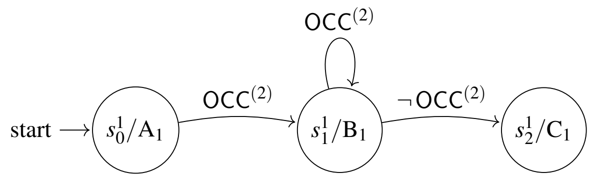
$$A_t \;=\; \lambda_i(s_{i,t}) \cup \lambda_j(s_{j,t}), \qquad t \geq 0,$$

and the successor states are determined by

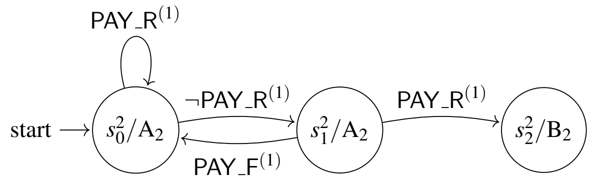
$$s_{i,t+1} = \delta_i(s_{i,t},\, \lambda_j(s_{j,t})), \qquad s_{j,t+1} = \delta_j(s_{j,t},\, \lambda_i(s_{i,t})).$$

Proof. Determinism ensures that for every product state $(s_{i,t}, s_{j,t})$ there is exactly one successor $(s_{i,t+1}, s_{j,t+1})$, determined by the mutual feedback of outputs. By induction on t, this yields a unique run of the product automaton starting from (s_i^0, s_j^0). Collecting the joint outputs at each step produces the word π, which is therefore unique. If the run stabilizes in a sink state with constant outputs, π is ultimately periodic and may be regarded as finite; otherwise π is infinite. $\square$

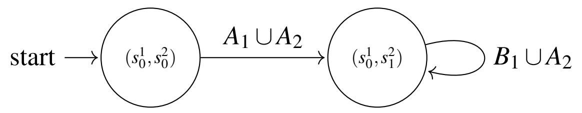
Example 14 (Product automaton sketch for $M_1^1 \times M_2^1$)**.** *We now project the two Moore machines M_1^1 (tenant) and M_2^1 (landlord) of Example 13 into their synchronous product $M_1^1 \times M_2^1$. Each state of the product records the joint output of both agents in that period, denoted by $\{A_1, A_2\}$, with $A_1 \in 2^{\Sigma_C^{(1)}}$ and $A_2 \in 2^{\Sigma_C^{(2)}}$. The initial state corresponds to (s_1^0, s_2^0); subsequent states reflect how both machines progress under synchronous feedback. Once a stable pair of states is reached, the product loops, generating the same joint output forever.*



(a) M_1^1 (tenant), with
$\mathrm{A}_1 = \{\mathsf{PAY\_R}^{(1)}\}$, $\mathrm{B}_1 = \{\mathsf{OCC}^{(1)}, \mathsf{PAY\_R}^{(1)}\}$, $\mathrm{C}_1 = \emptyset$.



(b) M_2^1 (landlord), with : $\mathrm{A}_2 = \{\mathsf{OCC}^{(2)}, \mathsf{PAY\_R}^{(2)}\}$, $\mathrm{B}_2 = \emptyset$



(c) Product automaton $M_1^1 \times M_2^1$, with states as pairs (s_i^1, s_j^2) and transitions labeled by the joint outputs.

Figure 3: Tenant's machine M_1^1, landlord's machine M_2^1, and their product automaton $M_1^1 \times M_2^1$. Each product state is labeled by the joint output $\lambda_1(s_i^1) \cup \lambda_2(s_j^2)$.

Concretely, substituting the outputs from Fig. 2, the product word begins as
$\{\mathsf{PAY\_R}^{(1)}, \mathsf{OCC}^{(2)}, \mathsf{PAY\_R}^{(2)}\}\ (\{\mathsf{OCC}^{(1)}, \mathsf{PAY\_R}^{(1)}, \mathsf{OCC}^{(2)}, \mathsf{PAY\_R}^{(2)}\})^{\omega}$.

4 The Two-Agents Collaborative Normative Logic TACNL

4.1 Syntax of TACNL

As summarized in Fig. 4, the syntax of TACNL is organized into three blocks: regular expressions, literals, and contracts.

Figure 4: Syntax of TACNL

Given a collaboration alphabet Σ_C and let $\Sigma := \Sigma_C^{(1)} \cup \Sigma_C^{(2)}$ be the tagged action alphabet. With: $a \in \Sigma_C$ (collaboration action), $p \in \{1,2\}$ (party). $A \in 2^{\Sigma}$ (party tagged action set), $n \in \mathbb{N}^*$ (non-zero natural number). The syntax of TACNL is defined inductively via the following grammar:

Regular expressions	re	$::=$	A	[tagged action set]
		$\mid$	Γ	[trump card]
		$\mid$	ε	[empty word]
		$\mid$	$\emptyset$	[empty set of actions]
		$\mid$	$(\mathsf{re} \mid \mathsf{re})$	[union]
		$\mid$	$\mathsf{re} \cdot \mathsf{re}$	[concatenation]
		$\mid$	re^n	[n-repetition]
		$\mid$	re^+	[Kleene plus]
Literals	ℓ	$::=$	$\mathbf{O}_p(a)$	[obligation]
		$\mid$	$\mathbf{F}_p(a)$	[prohibition]
		$\mid$	$\mathbf{P}_p(a)$	[permission]
		$\mid$	$\top$	[valid]
		$\mid$	$\bot$	[invalid]
Contracts	C	$::=$	ℓ	[literal]
		$\mid$	$C \wedge C$	[conjunction]
		$\mid$	$C;C$	[sequence]
		$\mid$	$C \blacktriangleright C$	[reparation]
		$\mid$	$\langle\langle re \rangle\rangle C$	[triggered]
		$\mid$	$\lceil re \rceil C$	[guarded]
		$\mid$	C^n	[n-repetition]
		$\mid$	$\mathbf{Rep}(C)$	[infinite-repetition]

Regular expressions (*re*). This block specifies *when* clauses apply by describing patterns over monthly positions. An *atom* is a tagged action-set $A \subseteq \Sigma$ stating what the two parties did in a month. Complex expressions are formed with *union* $(re \mid re)$, *con-*

catenation $(re \cdot re)$ for next-month sequencing, *power* re^n (exactly n repetitions), and *Kleene plus* re^+ (one or more repetitions). The *wildcard* Γ is the union of all $A \subseteq \Sigma$ used to skip a position, and $\emptyset$ denotes the *empty set of actions*. These constructs enable patterns such as "a repair is requested this month" (an atom), "after any number of months if Agent 1 asks for termination $(\Gamma \cdot)$, or "three consecutive months" $(re)^3$.

Literals (ℓ). This block provides the primitive deontic statements for a single month: *obligation* $\mathbf{O}_p(a)$, *prohibition* $\mathbf{F}_p(a)$, and *power* $\mathbf{P}_p(a)$, plus the constants *valid* $\top$ and *invalid* $\bot$. Here $p \in \{1,2\}$ identifies the party (tenant or landlord) and $a \in \Sigma_C$ is a collaboration action. Intuitively, literals say *what* is required/forbidden/allowed of *whom*, independently of timing.

Contracts (C). This block composes literals into full specifications using: *conjunction* $(C \wedge C)$ to combine requirements in the same time position; *sequence* $(C;C)$ for next-time progression; *reparation* $(C \blacktriangleright C')$ for contrary-to-duty fall-backs saying you are requested to perform C, if you fail you must conform to C' in the next time position; *triggered* clauses $\langle\langle re \rangle\rangle C$ that activate C when a pattern re occurs; *guarded* clauses $\lceil re \rceil C$ that encapsulate conditions under which conforming to a contract is no longer necessary; and *repetition* $\mathbf{Rep}(C)$ for repetitive occurrence of a contract. Together, these constructs could be used to capture the clauses of a contract, the conditions under which they are activated or terminated, and how the clauses relate to each other regarding reparations or the timing of their application. More specifically, the combination of repetition and a guarded contract can capture the notion of open-ended contracts. In the following example, we illustrate how we could capture our motivating example:

4.2 Illustrating the Encoding in TACNL for the Motivating Example

Example 15 (Encoding the rental clauses in TACNL). *We now illustrate how the rental agreement introduced in Example 1 can be systematically encoded in the* TACNL *syntax (see Fig. 4). We define the collaboration alphabet that captures all joint actions relevant to the contract:*

$$\Sigma_C = \{\mathsf{PAY\_R},\ \mathsf{PAY\_F},\ \mathsf{OCC},\ \mathsf{Notif\_R},\ \mathsf{Notif\_T},\ \mathsf{Maint\_P}\}.$$

Each element corresponds to a collaborative outcome: PAY_R *(rent payment),* PAY_F *(late fee payment),* OCC *(occupancy),* Notif_R *(tenant's repair request),* Notif_T *(termination notice), and* Maint_P *(landlord performing repair).*

The encoding proceeds clause by clause, following the contract structure:

- ***C1 (Tenant pays rent):*** *The tenant is obliged to pay the rent each month.*

$$C_1 := \mathbf{O}_1(\mathsf{PAY\_R}).$$

- ***C2 (Landlord guarantees occupancy):*** *The tenant gets the power to occupy the flat. Thus, the landlord is required not to interfere with the tenant's occupancy, encoded as a permission to allow the collaborative outcome* OCC*.*

$$C_2 := \mathbf{P}_1(\mathsf{OCC}).$$

- ***C3 (Late-payment reparation):*** *Clause C3 introduces a contrary-to-duty (CTD) structure: if the tenant fails to fulfill the primary obligation (C1), a compensatory obligation to pay the late fee arises. This relationship is encoded as a reparation construct:*
$$C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F}).$$

- ***C4 (Triggered repair request):*** *The tenant's request for repairs activates the landlord's duty to perform them within the following month. This is expressed using a triggered clause:*
$$C_4 := \langle\langle \{\mathsf{Notif\_R}^{(1)}\} \rangle\rangle \mathbf{O}_2(\mathsf{Maint\_P}).$$

- ***C5 (Termination and continuation):*** *The tenant may terminate the contract unilaterally by issuing a termination notice. After this notice, the contract's active obligations (rent, occupancy, repairs) persist for three additional months before ending. This behavior is captured with a guarded contract:*
$$C_5 := \lceil \Gamma^+ \cdot \{\mathsf{Notif\_T}^{(1)}\} \cdot \Gamma^3 \rceil (\, C_3 \wedge C_2 \wedge C_4 \,).$$

This step-by-step encoding shows how TACNL *integrates temporal regular patterns, deontic modalities, and event-triggered obligations in a single formalism. Clauses (C1–C5) together specify a full-cycle contract where collaborative actions such as payment, occupancy, repair, and termination are modeled as conditional and time-bounded obligations between the two agents.*

We move now to the first semantic definition for this logic, where we just care about whether a contract was satisfied.

5 The Notion of Tight Semantics

5.1 The Need for Tight Semantics

In TACNL , contracts integrate responsibilities across multiple agents and temporal dimensions, making timing integral to their interpretation. Each duty is associated with a specific moment, and failure to fulfill it punctually results in an immediate contractual shift: the violation is registered at a distinct point, after which reparation obligations are activated. Semantically, this necessitates partitioning a trace into a *pre-violation* prefix and a *post-violation* suffix, with the reparation component evaluated exclusively on the post-violation segment.

This partitioning must be exact. It is necessary to identify a unique earliest violation point, and correspondingly, a unique earliest satisfaction point, to ensure a single, unambiguous decomposition of observed behavior into segments occurring before and after the decisive instant. In the absence of such a unique boundary, reparations may be initiated prematurely, belatedly, or multiple times, leading to ambiguity in responsibility attribution. The prefix-based tight semantics introduced below are constructed to isolate these unique boundaries and thereby render the before-and-after evaluation well-defined.

This approach is termed *tight forward* semantics. The term *forward* indicates that each verdict is determined sequentially from left to right over prefixes, without reference to future events, and that contract progression follows the chronological order of the trace. The term *tight* signifies that the semantics are anchored at the first decisive instant, isolating a unique earliest satisfaction frontier and a unique earliest violation frontier, and classifying all strict extensions as post-frontier, thus precluding repeated triggering of the same responsibility. Thus, *tight forward* denotes a prefix-based, left-to-right evaluation with a uniquely defined division between pre- and post-frontier phases.

What goes wrong without tightness. If we only tag a prefix as "accepted" whenever it spells a word in the target language and keep tagging all longer extensions as "accepted" again, we *lose* the unique earliest acceptance point. This leads to (i) ambiguity about *when* credit is earned, (ii) potential "double counting" of compliance, and (iii) difficulty aligning guarded/triggered clauses with the moment they should switch on or off. Dually, labeling every failing extension as a fresh violation blurs *when* the duty was first broken.

Tiny illustration. Let $\Sigma = \{a,b\}$ and $L = \{a\}$ ("seeing a once is success"). Reading a at the first position should *decide* compliance then and there; the longer words $aa, ab, \ldots$ must be treated as *after* the decision, not as new acceptances. Conversely, reading b first fixes the earliest failure; $ba, bb, \ldots$ are merely *after* that failure.

What tight semantics will guarantee. Our five-valued, prefix-oriented view will:

- identify the *first acceptance* index (earliest satisfaction);
- identify the *first rejection* index (earliest violation);
- classify any strict extension *after* these frontiers as "post" acceptance/rejection;
- mark all prefixes that are still undecided but extendable to acceptance as *pre-eager*.

This yields determinacy (exactly one verdict per prefix), uniqueness of frontiers, and monotone *evolution* of verdicts along extensions—properties crucial for correctness, fairness, and auditable timing in contracts.

Building on this motivation, the following section introduces the language-theoretic operators and automata constructions that establish these frontiers and subsequently define the tight five-valued semantics.

5.2 From Language Membership to Tight Prefix

In this subsection, we generalize the construction of tight semantics to *any regular language* $L \subseteq \Sigma^*$. To do so, we must distinguish between the classical notion of *static language membership* and the requirements of *behaviour evaluation*.

Standard language theory evaluates a word w holistically: w is either inside or outside L. From the normative, behaviour-oriented point of view, we instead ask whether the desired behaviour has *already* been achieved on some prefix. This requires a prefix-sensitive notion of evaluation together with *eagerness*: we must identify the *first* prefix at which the behaviour becomes satisfied or becomes impossible. This viewpoint diverges from plain set membership. For example, if $L = \{a\}$, the word aa is not in L (it is rejected by language membership). However, for behavioural monitoring, the prefix a already establishes success; the second a is merely an irrelevant extension, not a failure.

Our objective is to formalize this shift by partitioning Σ^* into regions that isolate these *boundaries of decision*, explicitly resolving conflicts between "bad words" and "extensions of good words" in favor of the latter.

5.2.1 Topological Boundaries

We begin by identifying the candidate boundaries using standard topological operators on strings: the viable prefixes and the minimal evidence for membership or exclusion.

Definition 14 (Closures and Frontiers)**.** *For a regular language $L \subseteq \Sigma^*$:*

1. *The* ***Prefix Closure*** *($\mathsf{PCl}(L)$) is the set of all prefixes of words in L.*
2. *The* ***Bad Class*** *($\mathsf{BP}(L)$) is the complement of the closure (prefixes that can never lead to acceptance).*
3. *The* ***Minimal Frontier*** *($\mathsf{Min}(S)$) of a set is the set of its shortest elements.*

Applying the minimal frontier operator yields two sets of candidates:

- $\mathsf{Min}(L)$: The candidates for *Eager Acceptance* (shortest words in L).
- $\mathsf{Min}(\mathsf{BP}(L))$: The candidates for *Eager Rejection* (shortest words deviating from L).

5.2.2 The Priority of Acceptance

A rigorous definition of monitoring behavior requires resolving semantic overlaps between these frontiers. A word can be a minimal bad prefix while simultaneously extending a minimal accepted word (e.g., as noted, aa regarding $L = \{a\}$).

To ensure deterministic, forward-looking behavior, we enforce a **priority of acceptance**: Once a trace reaches the acceptance frontier $\mathsf{Min}(L)$, any further extension is classified as *irrelevant post-acceptance*, regardless of whether that extension technically belongs to $\mathsf{BP}(L)$.

Definition 15 (Canonical Semantic Partition)**.** *We define the five disjoint sets forming the partition of* Σ^* *as follows:*

$$\begin{aligned}
\mathsf{EA}(L) &:= \mathsf{Min}(L) && \textit{(Eager Acceptance)}\\
\mathsf{IAL}(L) &:= \mathsf{EA}(L)\,\Sigma^+ && \textit{(Irrelevant Acceptance)}\\
\mathsf{ER}(L) &:= \mathsf{Min}(\mathsf{BP}(L)) \setminus \mathsf{IAL}(L) && \textit{(Eager Rejection)}\\
\mathsf{IRL}(L) &:= \mathsf{ER}(L)\,\Sigma^+ && \textit{(Irrelevant Rejection)}\\
\mathsf{Pre}(L) &:= \mathsf{PCl}(\mathsf{EA}(L)) \setminus \mathsf{EA}(L) && \textit{(Pre-Verdict / Unknown)}
\end{aligned}$$

The critical operation here is the subtraction in $\mathsf{ER}(L)$. By removing $\mathsf{IAL}(L)$ from the bad frontier, we formally encode the shift from membership to monitoring: determining that a trace is "bad" is meaningful only if it has not already been declared "good."

Deconstructing regular language for tight behavior

Lemma 5. *For any* $L \subseteq \Sigma^*$*:*

1. $\mathsf{Min}(L) \subseteq \mathsf{PCl}(L)$*,* $\mathsf{Min}(\mathsf{BP}(L)) \subseteq \mathsf{BP}(L)$*, and* $\mathsf{Min}(L) \cap \mathsf{Min}(\mathsf{BP}(L)) = \emptyset$.
2. $\mathsf{PCl}(\mathsf{Min}(L)) = \{u \mid \exists m \in \mathsf{Min}(L) :\ u \preceq m\} = \big(\mathsf{PCl}(\mathsf{Min}(L)) \setminus \mathsf{Min}(L)\big) \,\dot\cup\, \mathsf{Min}(L)$.

Proof. (1) The inclusion $\mathsf{Min}(L) \subseteq \mathsf{PCl}(L)$ is immediate since every $m \in \mathsf{Min}(L)$ is itself a prefix of a word in L (namely m). Likewise $\mathsf{Min}(\mathsf{BP}(L)) \subseteq \mathsf{BP}(L)$ holds by definition of minimality within $\mathsf{BP}(L)$. Disjointness follows because $\mathsf{PCl}(L) \cap \mathsf{BP}(L) = \emptyset$.

(2) By definition of prefix-closure over the set of minimal acceptances. The decomposition into the disjoint union with $\mathsf{Min}(L)$ is immediate since $\mathsf{Min}(L) \subseteq \mathsf{PCl}(\mathsf{Min}(L))$ and the difference removes exactly the minimal elements. $\square$

Disjoint and complementary notation. We write for the *disjoint complementary union* of sets A, B, written $X = A \,\dot\cup\, B$. Thus $X = A \,\dot\cup\, B \,\dot\cup\, C$ asserts that A, B, C are pairwise disjoint, i.e, the intersection of any two different sets from $\{A, B, C\}$ is empty, and $X = A \cup B \cup C$.

Lemma 6 (Two canonical splits)**.** *For any* $L \subseteq \Sigma^*$*,*

$$\mathsf{PCl}(L) \;=\; \mathsf{PCl}(\mathsf{Min}(L)) \,\dot\cup\, \mathsf{IAL}(L), \qquad \mathsf{BP}(L) \;=\; \mathsf{SBad}(L) \,\dot\cup\, \mathsf{IRL}(L).$$

Proof sketch. For $u \in \mathsf{PCl}(L)$ pick $z \in L$ with $u \preceq z$ and let m be a shortest accepted prefix of z; then $m \in \mathsf{Min}(L)$. Either $u \preceq m$ (so $u \in \mathsf{PCl}(\mathsf{Min}(L))$) or $m \prec u$ (so $u \in m\Sigma^+ \subseteq \mathsf{IAL}(L)$). For the bad side, every $u \in \mathsf{BP}(L)$ has a unique shortest bad prefix $b \in \mathsf{Min}(\mathsf{BP}(L))$ with $b \preceq u$; if $b \notin \mathsf{IAL}(L)$ then either $u = b \in \mathsf{SBad}(L)$ or $u \in b\Sigma^+ \subseteq \mathsf{IRL}(L)$; if $b \in \mathsf{IAL}(L)$ it is assigned to acceptance-overshoot by convention. $\square$

Lemma 7 (Cross disjointness)**.** *For any* $L \subseteq \Sigma^*$*,*

$$\mathsf{PCl}(L) \cap \mathsf{BP}(L) = \emptyset, \qquad \mathsf{IRL}(L) \cap \mathsf{IAL}(L) = \emptyset, \qquad \mathsf{Min}(L) \cap \mathsf{IRL}(L) = \emptyset.$$

Five semantic regions

Definition 16 (Five semantic regions)**.** *For any $L \subseteq \Sigma^*$, define:*

$$\underbrace{\mathsf{PCl}(\mathsf{Min}(L)) \setminus \mathsf{Min}(L)}_{\text{pre-eager-verdict}} \,\dot\cup\, \underbrace{\mathsf{Min}(L)}_{\text{eager acceptance}} \,\dot\cup\, \underbrace{\mathsf{SBad}(L)}_{\text{eager rejection}} \,\dot\cup\, \underbrace{\mathsf{IAL}(L)}_{\text{irrelevant acceptance}} \,\dot\cup\, \underbrace{\mathsf{IRL}(L)}_{\text{irrelevant rejection}} .$$

Theorem 5.1 (Five-way partition of Σ^*)**.** *For every $L \subseteq \Sigma^*$, the space of all possible words could be decomposed into:*

$$\Sigma^* = \big(\mathsf{PCl}(\mathsf{Min}(L)) \setminus \mathsf{Min}(L)\big) \,\dot\cup\, \mathsf{Min}(L) \,\dot\cup\, \mathsf{SBad}(L) \,\dot\cup\, \mathsf{IAL}(L) \,\dot\cup\, \mathsf{IRL}(L).$$

Proof. From $\Sigma^* = \mathsf{PCl}(L) \,\dot\cup\, \mathsf{BP}(L)$ and Lemma 6,

$$\Sigma^* = \underbrace{\mathsf{PCl}(\mathsf{Min}(L)) \,\dot\cup\, \mathsf{IAL}(L)}_{\mathsf{PCl}(L)} \,\dot\cup\, \underbrace{\mathsf{SBad}(L) \,\dot\cup\, \mathsf{IRL}(L)}_{\mathsf{BP}(L)}.$$

Now split $\mathsf{PCl}(\mathsf{Min}(L))$ using Lemma 5(2). Cross disjointness follows from Lemma 7. □

Example 16 (Five regions illustration on a simple language)**.** *Let $\Sigma = \{a, b\}$ and $L = \{a\}$.*

$$\mathsf{PCl}(L) = \{\varepsilon, a\}, \qquad \mathsf{BP}(L) = \Sigma^* \setminus \{\varepsilon, a\} = \{b, aa, ab, ba, bb, \dots\},$$
$$\mathsf{Min}(L) = \{a\}, \qquad \mathsf{Min}(\mathsf{BP}(L)) = \{b, aa\}.$$

Tie-break and overshoots:

$$\mathsf{IAL}(L) = a\Sigma^+ = \{aa, ab, aaa, \dots\}, \qquad \mathsf{SBad}(L) = \{b\}, \qquad \mathsf{IRL}(L) = b\Sigma^+ = \{ba, bb, baa, \dots\}.$$

Five regions:

$$\underbrace{\{\varepsilon\}}_{\textit{pre-eager-verdict}} \,\dot\cup\, \underbrace{\{a\}}_{\textit{eager acceptance}} \,\dot\cup\, \underbrace{\{b\}}_{\textit{eager rejection}} \,\dot\cup\, \underbrace{a\Sigma^+}_{\textit{irrelevant acceptance}} \,\dot\cup\, \underbrace{b\Sigma^+}_{\textit{irrelevant rejection}} = \Sigma^*.$$

5.2.3 Tight Five-Valued Semantics

We now introduce a prefix-level semantics that takes values in the set $\mathbb{V}_5 = \{?, \top^t, \bot^t, \top^p, \bot^p\}$, corresponding respectively to: pre-eager verdict (undecided but extendable), eager acceptance (first satisfaction), eager rejection (first violation), irrelevant acceptance (post acceptance), and irrelevant rejection (post-rejection).

Shortcut notation. For a regular language $L \subseteq \Sigma^*$. For brevity, we fix:

$$\begin{array}{rcll}
\mathsf{Pre}(L) & := & \mathsf{PCl}(\mathsf{Min}(L)) \setminus \mathsf{Min}(L), & \text{(pre-eager verdict)},\\
\mathsf{EA}(L) & := & \mathsf{Min}(L), & \text{(eager acceptance)},\\
\mathsf{IAL}(L) & := & \mathsf{EA}(L)\,\Sigma^+, & \text{(irrelevant acceptance)},\\
\mathsf{ER}(L) & := & \mathsf{Min}(\mathsf{BP}(L)) \setminus \mathsf{IAL}(L), & \text{(eager rejection)},\\
\mathsf{IRL}(L) & := & \mathsf{ER}(L)\,\Sigma^+. & \text{(irrelevant rejection)}.
\end{array}$$

Definition 17 (Five-valued prefix semantics)**.** *Fix a regular language $L \subseteq \Sigma^*$. For any $u \in \Sigma^*$ define*

$$[\![u \models L]\!]_5 \;:=\; \begin{cases} ? & \text{if } u \in \mathsf{Pre}(L)\,, \\ \top^t & \text{if } u \in \mathsf{EA}(L)\,, \\ \bot^t & \text{if } u \in \mathsf{ER}(L)\,, \\ \top^p & \text{if } u \in \mathsf{IAL}(L)\,, \\ \bot^p & \text{if } u \in \mathsf{IRL}(L)\,. \end{cases}$$

Determinacy. By Theorem 5.1, the sets $\mathsf{Pre}(L)$, $\mathsf{EA}(L)$, $\mathsf{ER}(L)$, $\mathsf{IAL}(L)$, $\mathsf{IRL}(L)$ form a pairwise-disjoint and complete partition of Σ^*. Hence $[\![u \models L]\!]_5$ is well-defined and single-valued for every $u \in \Sigma^*$.

Theorem 5.2 (Monotonicity and Determinacy)**.** *The semantics satisfy the following stability properties for any $u \in \Sigma^*$:*

1. ***Stability of Verdicts:*** *If $[\![u \models L]\!]_5 \in \{\top^t, \top^p\}$, then for all extensions $v \succ u$, $[\![v \models L]\!]_5 = \top^p$. Analogously for rejection.*
2. ***Unique Frontier:*** *If $[\![u \models L]\!]_5 = \top^p$, there exists a unique strict prefix $u' \prec u$ such that $[\![u' \models L]\!]_5 = \top^t$.*
3. ***Determinism:*** *Since the partition in Theorem 5.1 is disjoint and complete, the semantics yields exactly one verdict for any input trace.*

Proof. As in the indexed setting, replace the slice $w[i,j]$ by the unindexed prefix u. Key facts: $\mathsf{Min}(L)$ and $\mathsf{Min}(\mathsf{BP}(L))$ are prefix-free; $\mathsf{IAL}(L) = \mathsf{Min}(L)\,\Sigma^+$; $\mathsf{IRL}(L) = \mathsf{SBad}(L)\,\Sigma^+$; and $\mathsf{ER}(L) = \mathsf{Min}(\mathsf{BP}(L)) \setminus \mathsf{IAL}(L)$. Items **(A)**–**(D)** follow immediately from these definitions. For **(E)**, take the unique shortest extensions from u into $\mathsf{EA}(L)$ and, when reachable, into $\mathsf{ER}(L)$; uniqueness is by minimality over $\mathbb{N}$ together with prefix-freeness. □

5.2.4 From Language Automaton to Tight Monitor Construction

This subsection explains how the deterministic language automaton obtained for a regular expression is lifted into a tight monitor. A language automaton recognises complete words of a regular expression, whereas a tight monitor must classify every prefix of every word into one of the five semantic regions. The transition structure of the automaton is therefore preserved, and only the output behaviour changes: each state receives a tight verdict according to the five-valued prefix semantics. This yields the tight monitor for regular expressions introduced in Definition 21.

We now recall the components used in this transformation and show how to convert the DFA into a Moore machine with tight verdicts.

Definition 18 (Five-region automata)**.** *Let $L \subseteq \Sigma^*$ be a regular language and let*

$$\mathcal{A}(L) = (Q, \Sigma, \delta, q_0, F) \quad \textit{with} \quad \mathcal{L}(\mathcal{A}(L)) = L$$

be a DFA for L. We define the following DFAs, all over the same alphabet Σ:

- $\mathcal{A}_{\mathrm{EA}}(L) := (Q_{\mathrm{EA}}, \Sigma, \delta_{\mathrm{EA}}, q^0_{\mathrm{EA}}, F_{\mathrm{EA}})$ *with* $\mathcal{L}(\mathcal{A}_{\mathrm{EA}}(L)) = \mathsf{EA}(L)$.
- $\mathcal{A}_{\mathrm{PRE}}(L) := (Q_{\mathrm{PRE}}, \Sigma, \delta_{\mathrm{PRE}}, q^0_{\mathrm{PRE}}, F_{\mathrm{PRE}})$ *with* $\mathcal{L}(\mathcal{A}_{\mathrm{PRE}}(L)) = \mathsf{Pre}(L)$.
- $\mathcal{A}_{\mathrm{IAL}}(L) := (Q_{\mathrm{IAL}}, \Sigma, \delta_{\mathrm{IAL}}, q^0_{\mathrm{IAL}}, F_{\mathrm{IAL}})$ *with* $\mathcal{L}(\mathcal{A}_{\mathrm{IAL}}(L)) = \mathsf{IAL}(L)$.
- $\mathcal{A}_{\mathrm{ER}}(L) := (Q_{\mathrm{ER}}, \Sigma, \delta_{\mathrm{ER}}, q^0_{\mathrm{ER}}, F_{\mathrm{ER}})$ *with* $\mathcal{L}(\mathcal{A}_{\mathrm{ER}}(L)) = \mathsf{ER}(L)$.
- $\mathcal{A}_{\mathrm{IRL}}(L) := (Q_{\mathrm{IRL}}, \Sigma, \delta_{\mathrm{IRL}}, q^0_{\mathrm{IRL}}, F_{\mathrm{IRL}})$ *with* $\mathcal{L}(\mathcal{A}_{\mathrm{IRL}}(L)) = \mathsf{IRL}(L)$.

These DFAs are obtained using the constructions described in the previous subsection (prefix-closure, complement, minimal frontiers, and right-ideal saturation).

Definition 19 (Five-region Moore machine). *Let $L \subseteq \Sigma^*$ be a regular language and let*

$$\mathcal{A}_{\mathrm{PRE}}(L),\, \mathcal{A}_{\mathrm{EA}}(L),\, \mathcal{A}_{\mathrm{ER}}(L),\, \mathcal{A}_{\mathrm{IAL}}(L),\, \mathcal{A}_{\mathrm{IRL}}(L)$$

be the five DFAs from Definition 18, with accepting sets $F_{\mathrm{PRE}}, F_{\mathrm{EA}}, F_{\mathrm{ER}}, F_{\mathrm{IAL}}, F_{\mathrm{IRL}}$. The five-region Moore machine *for L is the deterministic Moore machine*

$$\mathcal{M}_{5tight}(L) \;:=\; \big(S,\; s^0,\; \Sigma,\; \mathbb{V}_5,\; \delta,\; \lambda\big),$$

where

$$\begin{aligned}
S &:= Q_{\mathrm{PRE}} \times Q_{\mathrm{EA}} \times Q_{\mathrm{ER}} \times Q_{\mathrm{IAL}} \times Q_{\mathrm{IRL}},\\
s^0 &:= \big(q^0_{\mathrm{PRE}},\, q^0_{\mathrm{EA}},\, q^0_{\mathrm{ER}},\, q^0_{\mathrm{IAL}},\, q^0_{\mathrm{IRL}}\big),\\
\delta\big((p,e,r,a,\rho),\, \sigma\big) &:= \big(\delta_{\mathrm{PRE}}(p,\sigma),\; \delta_{\mathrm{EA}}(e,\sigma),\; \delta_{\mathrm{ER}}(r,\sigma),\; \delta_{\mathrm{IAL}}(a,\sigma),\; \delta_{\mathrm{IRL}}(\rho,\sigma)\big),\\
\lambda(p,e,r,a,\rho) &:= \begin{cases} \top^t & \textit{if } e \in F_{\mathrm{EA}},\\ \bot^t & \textit{if } r \in F_{\mathrm{ER}},\\ \top^p & \textit{if } a \in F_{\mathrm{IAL}},\\ \bot^p & \textit{if } \rho \in F_{\mathrm{IRL}},\\ ? & \textit{if } p \in F_{\mathrm{PRE}}. \end{cases}
\end{aligned}$$

We write $\mathcal{M}_{5tight}(L)$ as a function of L, since L uniquely determines all components of this Moore machine.

Example 17 (Compact five-valued Moore monitor). *The minimized five-output Moore machine for $L = \{a, ab, bb\}$ over $\Sigma = \{a, b\}$ produces $\mathbb{V}_5 = \{?, \top^t, \bot^t, \top^p, \bot^p\}$ according to the unique accepting component among the five regions* $\mathsf{Pre}(L), \mathsf{EA}(L), \mathsf{ER}(L), \mathsf{IAL}(L), \mathsf{IRL}(L)$.

Lemma 8 (Correctness of the tight-product Moore machine). *Let $L \subseteq \Sigma^*$ be regular and let $M_{5tight}(L)$ be as in Definition 19. For every $u \in \Sigma^*$,*

$$[\![u \models L]\!]_5 \;=\; \lambda\big(\delta^*(s^0, u)\big),$$

i.e., the output of $M_{5tight}(L)$ on input prefix u coincides with the five-valued semantics in Definition 17.

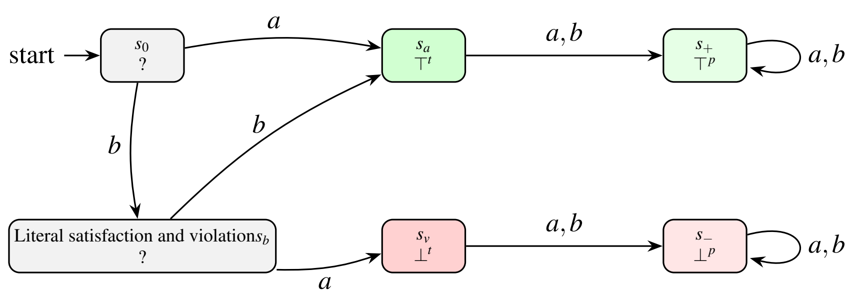


Figure 5: Minimized five-valued Moore machine for $L = \{a, ab, bb\}$. Each node shows its name and output verdict ($\mathbb{V}_5$). Green states denote satisfaction, red denotes violation, and gray denotes undecided. From s_0, input a or bb yields $\top^t$, input ba yields $\bot^t$, and further inputs move to the post-frontier verdicts $\top^p$ or $\bot^p$.

Proof. Let the five DFAs be

$$\mathcal{A}_{\text{PRE}}(L), \quad \mathcal{A}_{\text{EA}}(L), \quad \mathcal{A}_{\text{ER}}(L), \quad \mathcal{A}_{\text{IAL}}(L), \quad \mathcal{A}_{\text{IRL}}(L),$$

with recognized languages $\mathsf{Pre}(L)$, $\mathsf{EA}(L)$, $\mathsf{ER}(L)$, $\mathsf{IAL}(L)$, $\mathsf{IRL}(L)$, respectively. By construction of $M_{5\text{tight}}(L)$, after reading u the global state is

$$\delta^*(s^0, u) = \big(\delta^*_{\text{PRE}}(q^0_{\text{PRE}}, u),\ \delta^*_{\text{EA}}(q^0_{\text{EA}}, u),\ \delta^*_{\text{ER}}(q^0_{\text{ER}}, u),\ \delta^*_{\text{IAL}}(q^0_{\text{IAL}}, u),\ \delta^*_{\text{IRL}}(q^0_{\text{IRL}}, u)\big).$$

For each component DFA,

$$\begin{aligned}
u \in \mathsf{Pre}(L) &\iff \delta^*_{\text{PRE}}(q^0_{\text{PRE}}, u) \in F_{\text{PRE}},\\
u \in \mathsf{EA}(L) &\iff \delta^*_{\text{EA}}(q^0_{\text{EA}}, u) \in F_{\text{EA}},\\
u \in \mathsf{ER}(L) &\iff \delta^*_{\text{ER}}(q^0_{\text{ER}}, u) \in F_{\text{ER}},\\
u \in \mathsf{IAL}(L) &\iff \delta^*_{\text{IAL}}(q^0_{\text{IAL}}, u) \in F_{\text{IAL}},\\
u \in \mathsf{IRL}(L) &\iff \delta^*_{\text{IRL}}(q^0_{\text{IRL}}, u) \in F_{\text{IRL}}.
\end{aligned}$$

By Theorem 5.1, the five languages form a pairwise-disjoint, complete partition of Σ^*. Hence, for each u exactly one of the five memberships holds, and λ (by Definition 19) returns the unique verdict of Definition 17. Therefore $\lambda(\delta^*(s^0, u)) = [\![u \models L]\!]_5$. □

Proposition 5.3 (Linear-size Moore machine)**.** *Let $\mathcal{A}(L) = (Q, \Sigma, \delta, q_0, F)$ be a completed DFA for L, and let $M_{5tight}(L)$ be the tight five-valued Moore machine of Definition 19. Then the number of reachable states of $M_{5tight}(L)$ is linear in $|Q|$:*

$$\big|S_{\text{reach}}(M_{5tight}(L))\big| \;=\; \big|\text{Reach}(\mathcal{A}(L))\big| \;\le\; |Q|.$$

Proof. We argue directly from the constructions of the five DFAs used in $M_{5\text{tight}}(L)$. *(Same transition graph).* Starting from a completed DFA $\mathcal{A}(L) = (Q, \Sigma, \delta, q_0, F)$:

- The prefix-closure automaton $\mathsf{PC}(\mathcal{A}(L)) = (Q, \Sigma, \delta, q_0, \mathsf{Live})$ changes only the accepting set to Live (backward reachability to F).

- Its complement for bad-prefixes $\mathsf{BP}(\mathcal{A}(L)) = (Q, \Sigma, \delta, q_0, Q \setminus \mathsf{Live})$ flips finals, keeping (Q, Σ, δ).

- The minimal-frontier automaton $\mathsf{MinAut}(\mathcal{A}(L)) = (Q, \Sigma, \delta, q_0, F_{\min})$ selects the first accepting BFS layer; again, only finals change.

- The right-ideal closures for $\mathsf{IAL}(L)$ and $\mathsf{IRL}(L)$ "forward-saturate" the corresponding finals (every state reachable by a nonempty path becomes final), without altering (Q, Σ, δ).

- $\mathsf{ER}(L) = \mathsf{Min}(\mathsf{BP}(L)) \setminus \mathsf{IAL}(L)$ is realized on the same (Q, Σ, δ, q_0) by taking $F_{\mathrm{ER}} := F_{\min(\mathrm{Bad})} \setminus F_{\mathrm{IAL}}$, since both operands already use (Q, Σ, δ, q_0).

Thus, *all five DFAs share the identical transition graph* (Q, Σ, δ) and differ only in their accepting sets.

(Diagonal runs in the product). Fix any $u \in \Sigma^*$. Because the underlying transition function is the same in all five DFAs, the unique run from q_0 on u ends in the *same* state $q = \delta^*(q_0, u)$ in each DFA. Therefore, in the product Moore machine $M_{5\mathrm{tight}}(L)$, the product state reached by u is necessarily diagonal: (q, q, q, q, q).

(Size bound). Let $\mathsf{Reach}(\mathcal{A}(L)) := \{\delta^*(q_0, u) \mid u \in \Sigma^*\}$ be the set of states reachable in $\mathcal{A}(L)$. The mapping

$$q \mapsto (q, q, q, q, q)$$

is a bijection between $\mathsf{Reach}(\mathcal{A}(L))$ and the set $S_{\mathrm{reach}}(M_{5\mathrm{tight}}(L))$ of product states reachable in the Moore machine (by the diagonal property above). Hence

$$\left|S_{\mathrm{reach}}(\mathcal{M}_{5\mathrm{tight}}(L))\right| = \left|\mathsf{Reach}(\mathcal{A}(L))\right| \leq |Q|.$$

Therefore, the number of reachable Moore states grows at most linearly with the size of $\mathcal{A}(L)$. □

Additionally, each component DFA is deterministic (and can be completed), so $\mathcal{M}_{5\mathrm{tight}}(L)$ itself is a deterministic Moore machine over Σ that emits exactly one verdict from $\mathbb{V}_5 = \{?, \top^t, \bot^t, \top^p, \bot^p\}$ for every processed prefix. The evolution of verdicts along any word follows from Theorem **??**: the output leaves ? once to either $\top^t$ or $\bot^t$, and then remains in the corresponding post phase $\top^p$ or $\bot^p$.

Summary This section developed a *correct-by-construction* toolkit that turns any regular language $L \subseteq \Sigma^*$ into the five semantic regions required by tight monitoring, and then into a single verdicting monitor. The workflow relies only on classical automata-theoretic constructions—completion, product, complement, and breadth-first search on the DFA graph—used exactly as standard.

Starting from a DFA $\mathcal{A}(L)$, we systematically build the prefix-closure automaton $\mathsf{PC}(\mathcal{A}(L))$, which recognizes $\mathsf{PCl}(L)$; its complement $\mathsf{BP}(\mathcal{A}(L))$, which recognizes $\mathsf{BP}(L)$; and the minimal-frontier automaton $\mathsf{MinAut}(\mathcal{A}(L))$, which recognizes $\mathsf{Min}(L)$. Two right-ideal saturations yield the post regions $\mathsf{IAL}(L)$ and $\mathsf{IRL}(L)$, and one language difference realizes the acceptance-first tie-break $\mathsf{ER}(L) := \mathsf{Min}(\mathsf{BP}(L)) \setminus \mathsf{IAL}(L)$.

Practically, the approach is modular: each region is an ordinary DFA; scalable: construct only the reachable part of products and minimize components; and reusable across specifications that share the same L. Conceptually, it aligns the linguistic requirements of normative systems, where norms enter into force at exact positions, with executable monitors whose decisions are fair (no premature verdicts) and final (no evolution to the opposite verdict). In short, standard automata technology, assembled carefully, delivers a monitor that is *correct by design*.

5.3 Illustration Through Regular Expressions from TACNL

This section demonstrates the process of defining monitors for regular expressions from TACNL . The procedure begins by specifying the language semantics of the regular expression, followed by a standard transformation into a deterministic automaton. Subsequently, the transformation described in the previous subsection is applied to construct the five tight semantic monitors for the formula. This subsection fixes the semantics of the regular expressions that act as temporal guards in TACNL .

5.3.1 Semantics for Regular Expressions

We work over the *letter alphabet*

$$\Gamma := 2^{\Sigma_C^{(1)} \cup \Sigma_C^{(2)}},$$

Definition 20 (Semantics of regular expressions). *The satisfaction relation for a regular expression* re*, written* $\pi \models_{\mathsf{re}} \mathsf{re}$*, is defined over a finite trace* $\pi = \langle A_0, \ldots, A_{n-1} \rangle \in \Gamma^*$*, where each letter* $A_i \in \Gamma$ *is a set of actions that occurred at period i. The relation is given inductively:*

$$\begin{array}{lll}
\pi \models_{\mathsf{re}} \mathsf{A} & \textit{iff} & |\pi| = 1 \textit{ and } \mathsf{A} \subseteq A_0, \\
\pi \models_{\mathsf{re}} \Gamma & \textit{iff} & |\pi| = 1, \\
\pi \models_{\mathsf{re}} \varepsilon & \textit{iff} & \pi = \varepsilon, \\
\pi \models_{\mathsf{re}} \emptyset & \textit{iff} & A_0 = \emptyset \textit{ and } |\pi| = 1, \\
\pi \models_{\mathsf{re}} \mathsf{re}_1 \mid \mathsf{re}_2 & \textit{iff} & (\pi \models_{\mathsf{re}} \mathsf{re}_1) \textit{ or } (\pi \models_{\mathsf{re}} \mathsf{re}_2), \\
\pi \models_{\mathsf{re}} \mathsf{re}_1 \cdot \mathsf{re}_2 & \textit{iff} & \exists k \leq n \leq |\pi| : \pi[0,k] \models_{\mathsf{re}} \mathsf{re}_1 \textit{ and } \pi[k+1,n] \models_{\mathsf{re}} \mathsf{re}_2, \\
\pi \models_{\mathsf{re}} \mathsf{re}^n & \textit{iff} & (\textit{if } n > 1 \textit{ then } \pi \models_{\mathsf{re}} re \cdot re^{n-1}) \textit{ and } (\pi \models_{\mathsf{re}} re \textit{ if } n = 1), \\
\pi \models_{\mathsf{re}} \mathsf{re}^+ & \textit{iff} & \exists n \geq 1 : \pi \models_{\mathsf{re}} \mathsf{re}^n.
\end{array}$$

We write $\mathcal{L}(\mathsf{re}) := \{\pi \in \Gamma^* \mid \pi \models_{\mathsf{re}} \mathsf{re}\}$ *for the language of* re.

Reading the clauses.

- **Atom** A**.** Matches exactly one period: $\pi = \langle A_0 \rangle$ with $\mathsf{A} \subseteq A_0$.
- **Wildcard** Γ**.** Matches any single period: $\pi = \langle A_0 \rangle$ for arbitrary $A_0 \in \Gamma$.
- **Empty word** ε**.** Matches only the empty trace: $\pi = \varepsilon$.

- **Empty-action letter $\emptyset$.** Matches the one-period trace with no actions: $\pi = \langle\emptyset\rangle$.
- **Union $(re_1 \mid re_2)$.** Holds iff at least one disjunct holds on the whole trace.
- **Sequencing $(re_1 \cdot re_2)$.** There is a split index k with $\pi[0,k] \models_{\mathsf{re}} re_1$ and $\pi[k{+}1,n] \models_{\mathsf{re}} re_2$ (both parts finite).
- **Fixed power re^n.** Iterated sequencing of re exactly n times: $re^1 \equiv re$; for $n > 1$, $\pi \models_{\mathsf{re}} re^n$ iff $\pi \models_{\mathsf{re}} re \cdot re^{n-1}$.
- **Kleene plus re^+.** Some positive iteration holds: $\exists n \geq 1$ with $\pi \models_{\mathsf{re}} re^n$.

All satisfaction is defined on *finite* traces, we show how to detect triggers and terminating conditions on regular expressions using the 5-valued semantics.

5.3.2 Automata Construction Matching the Denotational Semantics

We now give a concrete, standard pipeline that realizes the semantics of Definition 20 *exactly* by an automaton over the alphabet $\Gamma = 2^{\Sigma_C^{(1)} \cup \Sigma_C^{(2)}}$.

We build $\mathcal{A}_\varepsilon(re)$ by structural recursion on re, using the usual two distinguished states $(s_{\text{in}}, s_{\text{out}})$ per fragment and ε-transitions for wiring ([18, 13, 17]). The only twist is how we treat letters, since an atom A matches *any* Γ-letter X that *covers* A (Definition 20).

- **Atom $\mathsf{A} \subseteq \Gamma$:** create two states $p \to q$ and add, for *every* $X \in \Gamma$ with $\mathsf{A} \subseteq X$, a transition $p \xrightarrow{X} q$. This enforces “one period, with all actions in A present.”
- **Wildcard Γ:** create $p \xrightarrow{X} q$ for *all* $X \in \Gamma$.
- **Empty word ε:** create $p \xrightarrow{\varepsilon} q$.
- **Empty-action letter $\emptyset$:** create a single-letter fragment $p \xrightarrow{\{\emptyset\}} q$ (i.e., only the Γ-letter $\emptyset$).
- **Union $(re_1 \mid re_2)$:** build fragments for re_1 and re_2 with entries/exits (p_1, q_1) and (p_2, q_2). Add fresh p, q and wire $p \xrightarrow{\varepsilon} p_1$, $p \xrightarrow{\varepsilon} p_2$, $q_1 \xrightarrow{\varepsilon} q$, $q_2 \xrightarrow{\varepsilon} q$.
- **Sequencing $(re_1 \cdot re_2)$:** build (p_1, q_1) and (p_2, q_2), then add $q_1 \xrightarrow{\varepsilon} p_2$ and take (p_1, q_2) as entry/exit. This matches the split $\pi[0,k]$ and $\pi[k{+}1,n]$ in the semantics.
- **Fixed power re^n:** unroll as $re; \cdots ; re$ (n times). The base $re^1 \equiv re$.
- **Kleene plus re^+:** build (p_1, q_1) for re, then add $q_1 \xrightarrow{\varepsilon} p_1$ and take (p_1, q_1) as entry/exit. (At least one iteration is enforced by entering at p_1.)

Mark the global entry of the whole construction as initial, and the global exit as accepting. The resulting NFA accepts exactly $\mathcal{L}(re)$.

Stage 2 Determinization. Apply the standard subset construction with ε-closures to obtain a DFA $\mathcal{A}_D(re) = (Q, \Gamma, \delta, q_0, F)$ that recognizes the same language ([16, 13]).

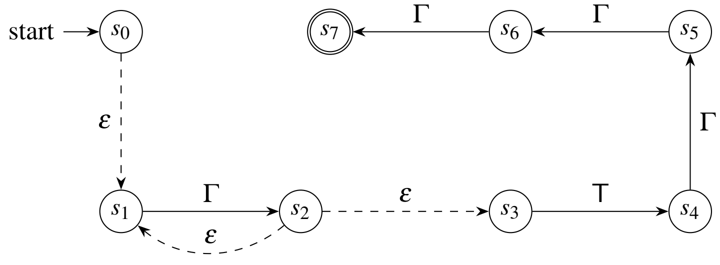
Correctness (sketch). By structural induction on *re*. The letter fragments implement exactly the one-step clauses (A, $*$, $\emptyset$), union and sequencing are the usual ε-wiring proofs, and re^n (unrolled) and re^+ (loop back from the exit) match the inductive clauses in Definition 20. Determinization and completion preserve language.

From $\mathcal{A}(re)$ to the tight layers. Follow the same steps from Definition .19:

Example 18 (End-to-end construction)**.** *Continuing from Example. 15, we demonstrate the automata construction for C_5:*

$$\begin{aligned} \mathsf{re}_{C_5} &:= \Gamma^+ \cdot \{\mathsf{Notif\_T}^{(1)}\} \cdot \Gamma^3, \\ \Gamma &:= 2^{\Sigma_C^{(1)} \cup \Sigma_C^{(2)}}, \\ \Sigma_C &:= \{\mathsf{PAY\_R},\ \mathsf{PAY\_F},\ \mathsf{OCC},\ \mathsf{Notif\_R},\ \mathsf{Notif\_T},\ \mathsf{Maint\_P}\}. \end{aligned}$$

Figure 6 depicts the Thompson-style ε-NFA for re_{C_5}, and Figure 7 shows its determinized and completed DFA.



Γ: any letter in 2^Σ $\quad$ T: set A with $\mathsf{Notif\_T}^{(1)} \in A$ $\quad$ dashed ε: wiring

Figure 6: Thompson-style ε-NFA for $\mathsf{re}_{C_5} = \Gamma^+ \cdot \{\mathsf{Notif\_T}^{(1)}\} \cdot \Gamma^3$ over $\Gamma = 2^\Sigma$. From s_0 we enter the Γ^+ block ($s_1 \xrightarrow{\Gamma} s_2$ with a back ε-loop to enforce "one or more" steps), then take a single T-labeled letter (the period that contains $\mathsf{Notif\_T}^{(1)}$), followed by exactly three arbitrary periods (three Γ transitions) to the accepting state s_7. Determinization and completion of this NFA yield a DFA that recognizes precisely the denotation of re_{C_5} in Definition 20.

5.3.3 From Language Automaton to Tight Monitor Construction

Definition 21 (Tight monitor construction for regular expressions)**.** *Let* re *be a regular expression over the alphabet* Γ *and let* $L := \mathcal{L}(\mathsf{re}) \subseteq \Gamma^*$ *be its language as in Definition 20. Let*

$$\mathcal{M}_{5tight}(L) = \left(S, s^0, \Gamma, \mathbb{V}_5, \delta, \lambda\right)$$

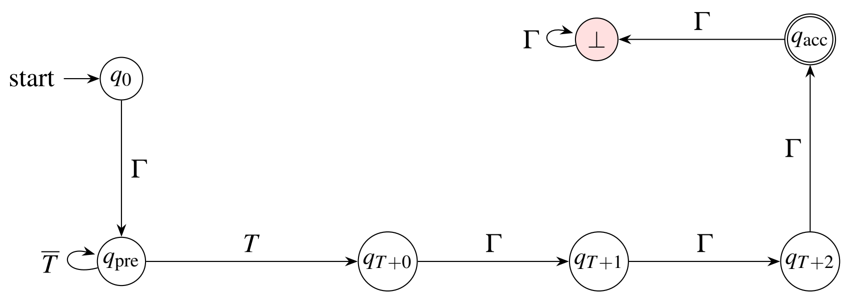


Figure 7: Determinized and completed DFA for $\mathsf{re}_{C_5} = \Gamma^+ \cdot \{\mathsf{Notif\_T}^{(1)}\} \cdot \Gamma^3$ over the alphabet $\Gamma = 2^\Sigma$. State q_{pre} collects the initial Γ^+ segment; transition on T begins the "+3 letters" counter ($q_{T+0} \to q_{T+1} \to q_{T+2} \to q_{\text{acc}}$). Any overrun moves to the sink.

be the five-region Moore machine for L from Definition 19. The tight satisfaction monitor *for* re *is this Moore machine:*

$$\mathsf{TSMC}_{\mathsf{re}}(\mathsf{re}) := \mathcal{M}_{5tight}(\mathcal{L}(\mathsf{re})).$$

By construction, for every prefix $u \in \Gamma^*$, the output of $\mathsf{TSMC}_{\mathsf{re}}(\mathsf{re})$ after reading u coincides with the tight five-valued semantics:

$$\lambda\big(\delta(s^0, u)\big) = [\![u \models \mathcal{L}(\mathsf{re})]\!]_5 .$$

Example 19 (Tight monitor for C_5)**.** *Continuing Example 18, Figure 8 shows the compact five-valued Moore monitor obtained by applying the tight monitor construction of Definition 21 to the regular expression* re_{C_5}.

6 Tight Forward Reasoning on Contract Compliance

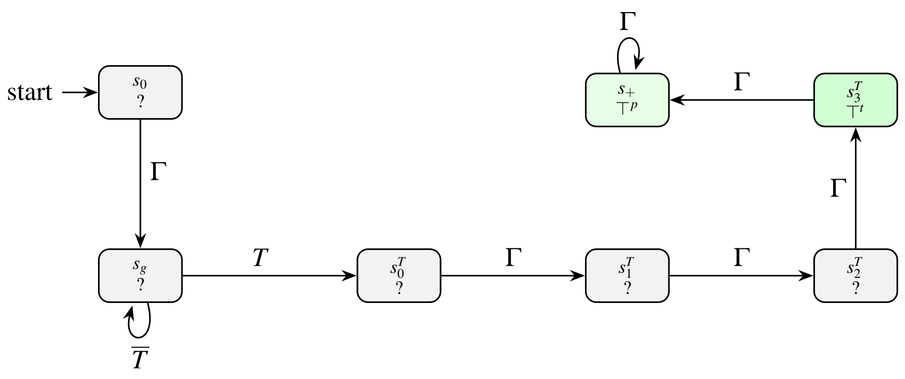
6.1 Denotational Semantics for Forward-Looking Tight Contract Satisfaction

Fix the tagged collaboration alphabet $\Sigma = \Sigma_C^{(1)} \cup \Sigma_C^{(2)}$ and the letter alphabet $\Gamma = 2^\Sigma$. A (finite or infinite) trace is $\pi = \langle A_0, A_1, \dots \rangle$ with $A_t \in \Gamma$. We write $|\pi| \in \mathbb{N} \cup \{\infty\}$ and use $\pi[0,k]$ for the prefix of length $k{+}1$ (inclusive).

Core tight judgements. We define two tight relations (inductively on the syntax of C):

$$\pi \models_{\top^t} C \quad \text{(tight satisfaction)}, \qquad \pi \models_{\perp^t} C \quad \text{(tight violation)}.$$

Intuitively, $\models_{\top^t}$ holds exactly at the *first prefix* where the contract becomes satisfied (acceptance frontier), and $\models_{\perp^t}$ holds exactly at the *first prefix* where it becomes violated (rejection frontier).



$T := \{A \in \Gamma \mid \mathsf{Notif\_T}^{(1)} \in A\}$, $\overline{T} := \Gamma \setminus T$, Γ = any letter in Γ.

Figure 8: Compact five-valued Moore monitor for $re_{C_5} = \Gamma^+ \cdot \{\mathsf{Notif\_T}^{(1)}\} \cdot \Gamma^3$. Outputs are shown inside the states ($\mathbb{V}_5$). All prefixes remain undecided (?) until the first usable termination event T appears, then exactly three more steps reach the tight satisfaction frontier $\top^t$, and further extensions yield the post-acceptance verdict $\top^p$. No $\bot^t/\bot^p$ states exist because no prefix is permanently rejecting.

Derived judgements. Using these frontiers, we finish defining the remaining relation from the 5 semantics:

Definition 22 (Post and pre satisfaction semantic definition)**.** *For a contract C and trace π the Pre satisfaction relation $\models_?$, the post satisfaction and violation relations, respectively $\models_{\top^p}$ and $\models_{\bot^p}$ are defined on the structure of the trace π and the tight satisfaction and violation relations:*

$$\begin{aligned}
\pi \models_? C &\iff \forall k < |\pi| : \neg\big(\pi[0,k] \models_{\top^t} C\big) \text{ and } \neg\big(\pi[0,k] \models_{\bot^t} C\big),\\
\pi \models_{\top^p} C &\iff \exists k < |\pi| : \pi[0,k] \models_{\top^t} C,\\
\pi \models_{\bot^p} C &\iff \exists k < |\pi| : \pi[0,k] \models_{\bot^t} C.
\end{aligned}$$

Each prefix is therefore either still undecided ($\models_?$), the unique first satisfaction ($\models_{\top^t}$), the unique first violation ($\models_{\bot^t}$), or strictly beyond one of these frontiers ($\models_{\top^p}$ or $\models_{\bot^p}$). These five regions are disjoint and jointly exhaustive.

Collapsed two-valued judgements. For downstream use (e.g., compliance checking), we collapse the five tight judgments into a two-valued view:

$$\begin{aligned}
\pi \models_\top C &\iff \big(\pi \models_{\top^t} C\big) \text{ or } \big(\pi \models_{\top^p} C\big),\\
\pi \models_\bot C &\iff \big(\pi \models_? C\big) \text{ or } \big(\pi \models_{\bot^t} C\big) \text{ or } \big(\pi \models_{\bot^p} C\big).
\end{aligned}$$

Exactly one of $\pi \models_\top C$ or $\pi \models_\bot C$ holds for every trace π and contract C. This conservative collapse treats undecided prefixes as *violating* (no premature acceptance) while still preserving the tight moment of satisfaction.

Literal Tight Semantics

Literals ℓ are decided in a single synchronous step. We therefore interpret them on a single event word $\langle A\rangle$ with $A \in \Gamma$ (the set of actions that occurred in one period). Literals are given for party $p = 1$; the case $p = 2$ is symmetric (by swapping $(\cdot)^{(1)}$ and $(\cdot)^{(2)}$) in the semantics. Intuitively, for party 1: (i) an *obligation* $\mathbf{O}_1(a)$ requires the joint execution of $a^{(1)}$ and $a^{(2)}$; (ii) a *prohibition* $\mathbf{F}_1(a)$ is satisfied precisely when that joint execution does not occur; and (iii) a *power* $\mathbf{P}_1(a)$ requires that whenever party 1 attempts $a^{(1)}$, party 2 simultaneously supports it with $a^{(2)}$.

Definition 23 (Literal tight satisfaction)**.** *Given a single event word $\langle A\rangle$ with $A \in \Gamma$, the tight semantics of literals for party $p = 1$ are:*

Tight satisfaction on $\langle A\rangle$

$$\begin{aligned}
\langle A\rangle &\models_{\top^t} \top && \overset{\text{def}}{=} \textit{true}.\\
\langle A\rangle &\models_{\top^t} \bot && \overset{\text{def}}{=} \textit{false}.\\
\langle A\rangle &\models_{\top^t} \mathbf{O}_1(a) && \overset{\text{def}}{=} \{a^{(1)}, a^{(2)}\} \subseteq A.\\
\langle A\rangle &\models_{\top^t} \mathbf{F}_1(a) && \overset{\text{def}}{=} \{a^{(1)}, a^{(2)}\} \not\subseteq A.\\
\langle A\rangle &\models_{\top^t} \mathbf{P}_1(a) && \overset{\text{def}}{=} \textit{if } (a^{(1)} \in A) \textit{ then } (a^{(2)} \in A).
\end{aligned}$$

Tight violation on $\langle A\rangle$

$$\begin{aligned}
\langle A\rangle &\models_{\bot^t} \top && \overset{\text{def}}{=} \textit{false}.\\
\langle A\rangle &\models_{\bot^t} \bot && \overset{\text{def}}{=} \textit{true}.\\
\langle A\rangle &\models_{\bot^t} \mathbf{O}_1(a) && \overset{\text{def}}{=} \{a^{(1)}, a^{(2)}\} \not\subseteq A.\\
\langle A\rangle &\models_{\bot^t} \mathbf{F}_1(a) && \overset{\text{def}}{=} \{a^{(1)}, a^{(2)}\} \subseteq A.\\
\langle A\rangle &\models_{\bot^t} \mathbf{P}_1(a) && \overset{\text{def}}{=} (a^{(1)} \in A) \textit{ and } (a^{(2)} \notin A).
\end{aligned}$$

The case $p = 2$ is symmetric.

Example 20 (Literal satisfaction and violation)**.** *Let $A = \{a^{(1)}, a^{(2)}, b^{(2)}\}$ be the joint actions in one period. Then*

$$\langle A\rangle \models_{\top^t} \mathbf{O}_1(a), \qquad \langle A\rangle \models_{\top^t} \mathbf{P}_1(a), \qquad \langle A\rangle \models_{\top^t} \mathbf{F}_2(b).$$

Let $A' = \{a^{(1)}, b^{(1)}, b^{(2)}\}$. Then

$\langle A'\rangle \models_{\bot^t} \mathbf{P}_1(a)$ *(since $a^{(2)} \notin A'$),* $\qquad \langle A'\rangle \models_{\top^t} \mathbf{F}_2(a)$ *(no joint $a^{(1)}$ and $a^{(2)}$ occurs).*

Also, $\langle A'\rangle \models_{\top^t} \mathbf{P}_2(a)$ holds vacuously since $a^{(2)} \notin A'$.

Two-event trace. *Consider $\langle A, A' \rangle$. Since literals are decided at the first letter, the overall collapsed verdict follows from $\langle A \rangle$ are post satisfaction or post violation:*

$$\begin{aligned}\langle A, A' \rangle &\models_{\top^p} \mathbf{O}_1(a),\\ \langle A, A' \rangle &\models_{\top^p} \mathbf{P}_1(a),\\ \langle A, A' \rangle &\models_{\bot^p} \mathbf{P}_2(b).\end{aligned}$$

Binary Contract Operators Tight Semantics

Binary contract operators combine two contracts into structured compositions that capture parallel, sequential, or conditional behavior. In TACNL we use

$$\mathsf{op} \in \{\wedge, ;, \blacktriangleright\},$$

where $\wedge$ enforces *both* components, ; demands *first C then C'*, and $\blacktriangleright$ means "if C fails, *repair* with C'."

Reading guide (tight view). All clauses below are tight: they identify the *first decisive point* where satisfaction or violation becomes determined. For conjunction, the decisive point for satisfaction is the latter of the two individual successes; for violation, the first conjunct that fails. For sequencing, a split index k witnesses that C succeeds before C' is checked. Reparation, on the other hand, requires that either C succeeds directly, or, at the first tight violation of C, the repair C' must succeed on the remainder.

Definition 24 (Binary Contract Operators)**.** *Let π be a finite trace over $\Gamma = 2^\Sigma$ with $s = |\pi|$, and let $k, k', k'' \in [0, s]$. By π_k we denote the prefix $\pi[0,k]$, and by π^k the suffix $\pi[k, |\pi|]$.*

Conjunction $(C \wedge C')$

$$\pi \models_{\top^t} C \wedge C' \overset{\mathsf{def}}{=} \exists k', k'' : \pi_{k'} \models_{\top^t} C \textit{ and } \pi_{k''} \models_{\top^t} C' \textit{ and } |\pi| = \max(k', k''),$$

$$\pi \models_{\bot^t} C \wedge C' \overset{\mathsf{def}}{=} (\pi \models_{\bot^t} C \textit{ or } \pi \models_{\bot^t} C') \textit{ and } \forall k' : \neg(\pi_{k'} \models_{\bot^t} (C \wedge C')),$$

Sequence $(C \,;\, C')$

$$\pi \models_{\top^t} C;C' \overset{\mathsf{def}}{=} \exists k : \pi_k \models_{\top^t} C \textit{ and } \pi^{k+1} \models_{\top^t} C',$$

$$\pi \models_{\bot^t} C;C' \overset{\mathsf{def}}{=} \pi \models_{\bot^t} C \textit{ or } \exists k : \pi_k \models_{\top^t} C \textit{ and } \pi^{k+1} \models_{\bot^t} C',$$

Reparation $(C \blacktriangleright C')$

$$\pi \models_{\top^t} C \blacktriangleright C' \overset{\mathsf{def}}{=} \pi \models_{\top^t} C \textit{ or } \exists k : \pi_k \models_{\bot^t} C \textit{ and } \pi^{k+1} \models_{\top^t} C',$$

$$\pi \models_{\bot^t} C \blacktriangleright C' \overset{\mathsf{def}}{=} \exists k : \pi_k \models_{\bot^t} C \textit{ and } \pi^{k+1} \models_{\bot^t} C'.$$

Semantics summary. *Conjunction* succeeds once both parts succeed (possibly at different times); its decisive index is the latter of the two. It fails as soon as either part fails. *Sequence* requires a witness split k: first C succeeds on $[0,k]$, then C' on $[k+1, s]$. *Reparation* allows C' to take over at the first violation of C; overall success means either direct success of C or a violation-then-repair pattern.

Example 21 (Tight satisfaction and violation for $(C_2 \wedge C_3)$)**.** *We reuse the collaboration alphabet*

$$\Sigma_C = \{\mathsf{PAY\_R},\ \mathsf{PAY\_F},\ \mathsf{OCC},\ \mathsf{Notif\_R},\ \mathsf{Maint\_P}\},$$

and recall

$$C_2 := \mathbf{P}_1(\mathsf{OCC}), \qquad C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F}).$$

Tight satisfaction (the longest prefix). *Consider the trace*

$$\pi_{\mathit{sat}} = \langle A_0, A_1 \rangle, \qquad A_0 = \{\mathsf{OCC}^{(2)}\}, \quad A_1 = \{\mathsf{PAY\_F}^{(1)}, \mathsf{PAY\_F}^{(2)}\}.$$

Then $\langle A_0 \rangle \models_{\top^t} C_2$ *(vacuously, since* $\mathsf{OCC}^{(1)} \notin A_0$ *and no unsupported attempt occurs), and* $\pi_{\mathit{sat}} \models_{\top^t} C_3$ *(the rent was not paid at* $t{=}0$*, but the reparation clause succeeds at* $t{=}1$*). Hence, by conjunction,* $\pi_{\mathit{sat}} \models_{\top^t} (C_2 \wedge C_3)$ *at the longest decisive prefix.*

Tight satisfaction (the shortest prefix). *For the single-event trace*

$$\langle A'_0 \rangle \quad \textit{with} \quad A'_0 = \{\mathsf{OCC}^{(2)}, \mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\},$$

we have $\langle A'_0 \rangle \models_{\top^t} C_2$ *and* $\langle A'_0 \rangle \models_{\top^t} \mathbf{O}_1(\mathsf{PAY\_R})$*, so both conjuncts hold at* $t{=}0$*:*

$$\langle A'_0 \rangle \models_{\top^t} (C_2 \wedge C_3), \quad \textit{and any extension } \langle A'_0, A_1 \rangle \textit{ yields } \models_{\top^p} (C_2 \wedge C_3).$$

Tight violation. *Now consider*

$$\pi_{\mathit{viol}} = \langle A_0, A_1 \rangle, \qquad A_0 = \{\mathsf{OCC}^{(1)}\}, \quad A_1 = \emptyset.$$

At $t{=}0$*,* $\langle A_0 \rangle \models_{\top^t} C_2$ *and* $\langle A_0 \rangle \models_{\perp^t} \mathbf{O}_1(\mathsf{PAY\_R})$*, while* $\langle A_0 \rangle \models_{?} C_3$ *since the reparation in* $C_3 = \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ *has not yet been tested. At* $t{=}1$*,* $\pi_{\mathit{viol}} \models_{\perp^t} C_3$ *(as* $\pi[0,0] \models_{\perp^t} \mathbf{O}_1(\mathsf{PAY\_R})$ *and* $\pi[1,1] \models_{\perp^t} \mathbf{O}_1(\mathsf{PAY\_F})$*). Thus, the overall violation arises from* C_3*, and by conjunction,* $\pi_{\mathit{viol}} \models_{\perp^t} (C_2 \wedge C_3)$*.*

This shows that $(C_2 \wedge C_3)$ *satisfies either immediately when both conjuncts hold, or later when a reparation compensates for a missed rent, while violation arises when both payment and its repair fail.*

Repetition Contracts Tight Semantics

Definition 25 (Repetition Contracts)**.** *Let* π *be a finite trace over the event alphabet* $\Gamma = 2^{\Sigma}$*, with* $s = |\pi|$ *and* $k \in [0, s-1]$ *and* n *a strictly positif natural number* $n \in \mathbb{N}^*$*. We refer by* π^k *the suffix* $\pi[k, |\pi|]$*, and by* π_k *to the prefix* $\pi[0,k]$*. The semantics for repetition contracts are inductively defined as follows:*

$$\begin{aligned}
\pi \models_{\top^t} C^n &\overset{\text{def}}{=} \big(n > 1 \Rightarrow \pi \models_{\top^t} C; C^{n-1}\big) \textit{ and } \big(n = 1 \Rightarrow \pi \models_{\top^t} C\big),\\
\pi \models_{\perp^t} C^n &\overset{\text{def}}{=} \big(\pi \models_{\perp^t} C\big) \textit{ or } \big(\exists k, m < n : \pi_k \models_{\top^t} C^m \textit{ and } \pi^{k+1} \models_{\perp^t} C\big),\\
\pi \models_{\top^t} \mathbf{Rep}(C) &\overset{\text{def}}{=} \textit{false},\\
\pi \models_{\perp^t} \mathbf{Rep}(C) &\overset{\text{def}}{=} \exists n : \pi \models_{\perp^t} C^n.
\end{aligned}$$

Intuition. Repetition contracts express the iterative enforcement of a subcontract. The finite form C^n requires C to hold n times in sequence, each instance starting immediately after the previous one completes. The satisfaction condition unfolds recursively: a trace satisfies C^n if it can be decomposed into a prefix where C holds, followed by a suffix that satisfies C^{n-1}. A violation occurs either when the first occurrence of C fails, or when some later repetition cannot be fulfilled after a previously satisfied segment. Hence, C^n behaves as a *sequential chain* of responsibilities and rights, and any broken link invalidates the entire chain.

The infinite form $\mathbf{Rep}(C)$ captures *unbounded repetition*. Since finite traces cannot exhibit infinite iteration, $\mathbf{Rep}(C)$ is never fully satisfied (false under tight semantics); it is only meaningful with respect to violation: a trace violates $\mathbf{Rep}(C)$ once it violates one of its finite unfolding C^n. Intuitively, $\mathbf{Rep}(C)$ models *renewable or continuing* contracts such as subscriptions or recurring payments, where each cycle restarts the same normative condition indefinitely.

Contracts-Regular Expression Binary Operator Semantics

Contracts guarded by regular expressions specify that a normative condition becomes active only after the trace matches a given regular pattern. Such patterns, written *re*, are interpreted over the letter alphabet $\Gamma = 2^{\Sigma}$ introduced above. They act as *temporal triggers* that delimit where an obligation, prohibition, or reparation clause starts to apply.

Two guarded forms are distinguished:

- The *triggered contract* $\langle\langle re\rangle\rangle C$, which activates C as soon as a prefix of the trace matches *re*.
- The *guarded contract* $\lceil re\rceil C$, which restricts C to hold only while the trace remains within the language induced by *re*.

The first captures temporal activation ("after the trigger, C must hold"), the second conditional persistence ("as long as *re* remains possible, C must hold").

Definition 26 (Triggered and Guarded Contracts)**.** *Let π be a finite trace over $\Gamma = 2^{\Sigma}$ and $k \in [0, |\pi|]$.*

$$\begin{aligned}
\pi \models_{\top^t} \langle\langle re\rangle\rangle C &\overset{\text{def}}{=} \pi \models_{\bot^t} re \text{ or } \big(\exists k : \pi_k \models_{\top^t} re \text{ and } \pi^{k+1} \models_{\top^t} C\big),\\
\pi \models_{\bot^t} \langle\langle re\rangle\rangle C &\overset{\text{def}}{=} \exists k : \pi_k \models_{\top^t} re \text{ and } \pi^{k+1} \models_{\bot^t} C,\\
\pi \models_{\top^t} \lceil re\rceil C &\overset{\text{def}}{=} \big(\pi \models_{\bot^t} re \text{ and } \pi \models_{Cl} C\big) \text{ or } \big(\pi \models_{Cl} re \text{ and } \pi \models_{\top^t} C\big),\\
\pi \models_{\bot^t} \lceil re\rceil C &\overset{\text{def}}{=} \pi \models_{Cl} re \text{ and } \pi \models_{\bot^t} C.
\end{aligned}$$

Where $\pi \models_{Cl} X$ abbreviates $(\pi \models_{?} X \text{ or } \pi \models_{\top^t} X)$.

Example 22 (Triggered and guarded contracts)**.** *Let the collaboration alphabet be*

$$\Sigma_C = \{\mathsf{PAY\_R},\ \mathsf{PAY\_F},\ \mathsf{OCC},\ \mathsf{Notif\_R},\ \mathsf{Notif\_T},\ \mathsf{Maint\_P}\}.$$

(a) Triggered contract. *Clause C_4 specifies that when the tenant requests a repair,the landlord must perform it within the following period:*

$$C_4 := \langle\langle \{\mathsf{Notif\_R}^{(1)}\} \rangle\rangle \mathbf{O}_2(\mathsf{Maint\_P}).$$

Tight satisfaction.

$$\pi_{\mathsf{sat}} = \langle A_0, A_1 \rangle, \qquad A_0 = \{\mathsf{Notif\_R}^{(1)}\}, \quad A_1 = \{\mathsf{Maint\_P}^{(1)}, \mathsf{Maint\_P}^{(2)}\}.$$

At t=0, the trigger $\mathsf{Notif\_R}^{(1)}$ occurs, activating the repair obligation. At t=1, the landlord performs $\mathsf{Maint\_P}^{(1,2)}$, thus $\pi_{\mathsf{sat}} \models_{\top^t} C_4$.

Tight violation.

$$\pi_{\mathsf{viol}} = \langle A_0, A_1 \rangle, \qquad A_0 = \{\mathsf{Notif\_R}^{(1)}\}, \quad A_1 = \emptyset.$$

The trigger fires at t=0, but the obligation is unfulfilled: $\pi_{\mathsf{viol}} \models_{\perp^t} C_4$.

(b) Guarded repetition. *To limit repetition to the occupancy period, combine guard and repetition:*

$$C_9 := \lceil *^+ ; \{\mathsf{Notif\_T}^{(1)}\} ; *^3 \rceil \mathbf{Rep}(\mathbf{O}_1(\mathsf{PAY\_R})).$$

The guard pattern $*^+;\{\mathsf{Notif\_T}^{(1)}\};*^3$ means "for any non-empty prefix up to the termination notice $\mathsf{Notif\_T}^{(1)}$, and for at most three additional steps afterward." Within this region, the obligation to pay rent repeats. Once $\mathsf{Notif\_T}^{(1)}$ occurs, the duty remains for three more periods, and the contract is satisfied at t=1+3=4.

Tight satisfaction.

$$\pi_{\mathsf{sat}} = \langle A_0, A_1, A_2, A_3, A_4 \rangle, \quad \begin{aligned} A_0 &= \{\mathsf{OCC}^{(1)}, \mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}, \\ A_1 &= \{\mathsf{Notif\_T}^{(1)}, \mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}, \\ A_2 = A_3 = A_4 &= \{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}. \end{aligned}$$

The guard is satisfied through t=4, hence $\pi_{\mathsf{sat}} \models_{\top^t} C_9$.

Tight violation.

$$\pi_{\mathsf{viol}} = \langle A_0, A_1, A_2, A_3, A_4 \rangle, \quad \begin{aligned} A_0 &= \{\mathsf{OCC}^{(1)}, \mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}, \\ A_1 &= \{\mathsf{Notif\_T}^{(1)}, \mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}, \\ A_2 &= \{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}, \\ A_3 &= \emptyset, \\ A_4 &= \{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}. \end{aligned}$$

A missing payment at t=3 breaks the repetition duty while the guard still holds, so $\pi_{\mathsf{viol}} \models_{\perp^t} C_9$.

Coherence of the Forward-Looking Contract Satisfaction Semantics

Coherence requires that for any fixed contract and trace, there is never more than one decisive verdict. A trace cannot both tightly satisfy and tightly violate the same contract on different prefixes, since this would yield two incompatible outcomes for a single execution. Forward semantics must therefore rule out situations where tight satisfaction appears on one prefix and tight violation appears on another prefix of the same trace. Ensuring this exclusion makes the decisive point unique, which is required to justify every verdict from $\mathbb{V}_5$. The next lemma states this exclusion precisely by showing that the two frontiers cannot arise on distinct prefixes of the same trace.

Lemma 9 (Mutual prefix exclusion tight satisfaction and violation)**.** *For every contract C in* TACNL *and every finite trace π, the tight satisfaction and tight violation forward semantics are mutually exclusive, that is:*

1. No earlier tight violation at or after tight satisfaction.
   if $\pi \models_{\top^t} C$ *then* $\nexists\, j < |\pi| : \pi[0,j] \models_{\perp^t} C$
2. No earlier tight satisfaction at or after tight violation.
   if $\pi \models_{\perp^t} C$ *then* $\nexists\, j < |\pi| : \pi[0,j] \models_{\top^t} C$

Proof sketch. By structural induction on the syntactical structure of C.

Base case: literals. By Def. 23, a literal is decided on a single letter: $\langle A\rangle \models_{\top^t} \ell$ iff the letter constraint holds, and $\langle A\rangle \models_{\perp^t} \ell$ iff it does not. These are complements on that step, so the two implications are immediate, and uniqueness follows.

Inductive hypotheses. Assume the theorem holds for subcontracts as needed below. We use:

$$\begin{aligned} &(\text{IH-}C\text{-sat}) && \forall\pi\,\big(\pi \models_{\top^t} C \Rightarrow \forall j < |\pi| : \neg(\pi[0,j] \models_{\perp^t} C)\big),\\ &(\text{IH-}C\text{-viol}) && \forall\pi\,\big(\pi \models_{\perp^t} C \Rightarrow \forall j < |\pi| : \neg(\pi[0,j] \models_{\top^t} C)\big), \end{aligned}$$

and similarly (IH-C'-sat) and (IH-C'-viol) when a second operand C' is present; for regex guards *re* we use the same two clauses with *re* in place of C.

Conjunction $C \wedge C'$. By Def. 24, tight satisfaction requires first successes at some k, k' with decisive index $j^\star = \max\{k,k'\}$. For every $j < j^\star$, either $j < k$ or $j < k'$ holds, hence by (IH-C-sat) and (IH-C'-sat) neither $\pi[0,j] \models_{\perp^t} C$ nor $\pi[0,j] \models_{\perp^t} C'$ holds. Since a tight violation of a conjunction is a tight violation of the conjunction, no $j < j^\star$ violates $C \wedge C'$. This proves the first implication. For the second, if some prefix tightly violates a conjunct, then by (IH-C-viol) or (IH-C'-viol) no earlier prefix tightly satisfies that conjunct, hence, no earlier prefix tightly satisfies the conjunction.

Sequence $C;C'$. By Def. 24, tight satisfaction needs a split k with $\pi[0,k] \models_{\top^t} C$ and $\pi[k{+}1,|\pi|] \models_{\top^t} C'$. For any $j \le k$, (IH-C-sat) forbids $\pi[0,j] \models_{\perp^t} C$; for any $j > k$, (IH-C'-sat) applied to the suffix forbids $\models_{\perp^t} C'$ before its own decisive point. A tight violation of $C;C'$ before satisfaction is either a violation of C before k or a violation of C' after k, both excluded. The dual implication follows from (IH-C-viol) and (IH-C'-viol).

Reparation $C \blacktriangleright C'$. By Def. 24, either C succeeds, or else at the first tight violation index k of C the repair C' must succeed on π^{k+1}. In the first branch (IH-C-sat), it excludes earlier violations. In the second branch, (IH-C-viol) gives minimal property of the failure point of C, and (IH-C'-sat) on the suffix excludes earlier failure of the composite before its tight success. The dual implication is symmetric, using (IH-C-viol) and (IH-C'-viol).

Finite repetition C^n. Unfold $C^n \equiv C;(C^{n-1})$ and argue by a secondary induction on n, using the sequence case and the induction hypotheses for C and C^{n-1}.

Unbounded repetition $\mathbf{Rep}(C)$. Under tight semantics $\mathbf{Rep}(C)$ never tightly satisfies and tightly violates iff some finite unrolling C^m tightly violates. The two implications reduce to the finite case above.

Triggered $\langle re \rangle C$. By Def. 26, either re is violated and the contract tightly satisfies vacuously, or there is a first k with $\pi_k \models_{\top^t} re$ and then the suffix must satisfy C. In the vacuous branch, (IH-re-viol) forbids any earlier tight satisfaction of re, so there is no earlier tight violation of the composite. In the active branch, the first match index k is minimal by (IH-re-sat); before k the composite is undecided, and after k we apply (IH-C-sat)/(IH-C-viol) on the suffix to obtain both implications.

Guarded $[re]C$. While π *closes re* (that is, π is in the open region for re), any tight or post failure of C yields a tight or post failure of the composite. Once re becomes impossible, the composite satisfies provided C has not failed. Combine (IH-re-sat) and (IH-re-viol) with (IH-C-sat) and (IH-C-viol), and the guarded case table in Def. 26, to derive the two implications.

All constructors preserve the two "no-backtrack" properties; hence, the claim holds for all C. $\square$

Theorem 6.1 (Consistency of the forward looking 5 tight semantics)**.** *The five forward satisfaction relations* $\{\models_?, \models_{\top^t}, \models_{\bot^t}, \models_{\top^p}, \models_{\bot^p}\}$ *for* TACNL *are pairwise disjoint and jointly exhaustive.*

Proof. By Lemma 9, there is *at most one* tight frontier: if $\pi \models_{\top^t} C$ then no prefix tightly violates C, and if $\pi \models_{\bot^t} C$ then no prefix tightly satisfies C. Hence, there exists at most one k with $\pi[0,k] \models_{\top^t} C$ and at most one k' with $\pi[0,k'] \models_{\bot^t} C$, and these cannot coexist.

By Definition 22, every prefix is classified by whether a frontier has not yet appeared ($\models_?$), is exactly at the first satisfaction or violation ($\models_{\top^t}$ or $\models_{\bot^t}$), or strictly follows the unique frontier ($\models_{\top^p}$ or $\models_{\bot^p}$). The lemma's uniqueness and non-coexistence ensure these five regions are pairwise disjoint. They are jointly exhaustive, since every prefix is either before any frontier, at the (unique) frontier, or strictly after it. Thus, $\{\models_?, \models_{\top^t}, \models_{\bot^t}, \models_{\top^p}, \models_{\bot^p}\}$ forms a partition of prefixes, proving the claim. $\square$

6.2 Monitor Construction for Tight Contract Satisfaction

Definition 27 (Tight satisfiaction monitor)**.** *The* tight satisfaction monitor*, written $\mathcal{M}^{TS}$, is a Moore machine whose output alphabet is the five-valued verdict set $\mathbb{V}_5$. Formally,*

$$\mathcal{M}^{TS} = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5),$$

where:

1. *The output alphabet formed by 5 letters is*

$$\mathbb{V}_5 = \{?, \top^t, \bot^t, \top^p, \bot^p\}.$$

2. *Q is the set of states and $q_0 \in Q$ is the initial state,*
3. *$\Gamma = 2^\Sigma$ is the input event alphabet,*
4. *$\delta : Q \times \Gamma \to Q$ is the transition function,*
5. *$\lambda_5 : Q \to \mathbb{V}_5$ is the state output function.*

The next definition introduces the construction for any contract into its tight satisfaction monitor. The construction is a function that maps each contract C to a tight satisfaction monitor that enforces it. The construction proceeds by structural induction on the syntax of C, and each operator in TACNL is matched by a corresponding monitor combination operator. Regular expression guards and triggers rely on the tight satisfaction monitor $\mathsf{TSMC}(\mathsf{re})$ introduced earlier in Definition 21.

Definition 28 (Tight Satisfaction Monitor Construction)**.** *The* tight satisfaction monitor construction *is a function defined on* TACNL*, written* $\mathsf{TSMC}(C)$*, that returns the tight satisfaction monitor for a contract C in* TACNL *. It is defined inductively on the structure of C:*

$$\mathsf{TSMC}(C) := \begin{cases} \mathsf{TSMC}_{lit}(\ell) & \textit{if } C = \ell, \\ \mathsf{TSMC}_{\wedge}\big(\mathsf{TSMC}(C_1), \mathsf{TSMC}(C_2)\big) & \textit{if } C = C_1 \wedge C_2, \\ \mathsf{TSMC}_{;}\big(\mathsf{TSMC}(C_1), \mathsf{TSMC}(C_2)\big) & \textit{if } C = C_1 ; C_2, \\ \mathsf{TSMC}_{\blacktriangleright}\big(\mathsf{TSMC}(C_1), \mathsf{TSMC}(C_2)\big) & \textit{if } C = C_1 \blacktriangleright C_2, \\ \mathsf{TSMC}_{trig}(\mathsf{re}, C') & \textit{if } C = \langle\langle \mathsf{re} \rangle\rangle C', \\ \mathsf{TSMC}_{guard}(\mathsf{re}, C') & \textit{if } C = \lceil \mathsf{re} \rceil C', \\ \mathsf{TSMC}_{nrep}\big(n, \mathsf{TSMC}(C')\big) & \textit{if } C = (C')^n, \\ \mathsf{TSMC}_{Rep}\big(\mathsf{TSMC}(C')\big) & \textit{if } C = \mathbf{Rep}(C') . \end{cases}$$

Where the tight monitor construction for regular expressions $\mathsf{TSMC}(\mathsf{re})$ *is already defined in Definition 21.*

Construction for Literal Contracts

From Tight Semantics to 5-Valued Monitoring. The literal clauses above define one-step satisfaction and violation judgements for a single event word $\langle A \rangle$. We lift these clauses into a five-valued Moore machine whose outputs track the evolution of the tight verdicts over prefixes.

Definition 29 (Tight Satisfaction Monitor Construction for Literals)**.** *For a literal ℓ from* TACNL *, the* Tight Satisfaction monitor construction *for ℓ, written $\mathsf{TSMC}_{lit}(\ell)$ is defined as:*

$$\mathsf{TSMC}_{lit}(\ell) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5).$$

- *$Q = \{q_0, q_s, q_v, q_{ps}, q_{pv}\}$, with outputs $\lambda_5(q_0) = ?$, $\lambda_5(q_s) = \top^t$, $\lambda_5(q_v) = \bot^t$, $\lambda_5(q_{ps}) = \top^p$, $\lambda_5(q_{pv}) = \bot^p$.*
- *$\Gamma = 2^\Sigma$ is the event alphabet.*
- *$\delta : Q \times \Gamma \to Q$ is defined as follows:*

$$\text{(1) tight transition:} \quad \delta(q_0, A) = \begin{cases} q_s & \text{if } \langle A \rangle \models_{\top^t} \ell, \\ q_v & \text{if } \langle A \rangle \models_{\bot^t} \ell; \end{cases}$$

$$\text{(2) post transitions:} \quad \begin{array}{l} \delta(q_s, A) = q_{ps},\ \delta(q_v, A) = q_{pv}, \\ \delta(q_{ps}, A) = q_{ps},\ \delta(q_{pv}, A) = q_{pv}. \end{array}$$

Hence, for every atomic literal, the machine emits ? at the initial state, switches to $\top^t$ or $\bot^t$ at the next state by consuming the event forming $\langle A \rangle$, and then permanently outputs $\top^p$ or $\bot^p$ for all remaining events. This Moore representation is equivalent to the tight semantics of Definition 23 but refines it with explicit prefix continuity.

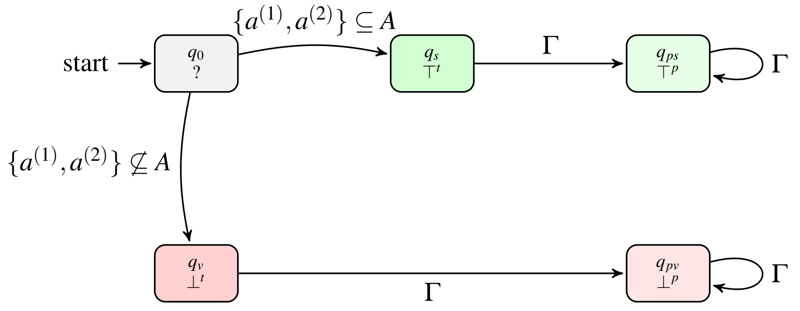


Figure 9: Compact 5-verdict Moore machine for the obligation literal $\mathbf{O}_1(a)$. Each node displays its internal state and the corresponding output verdict $\in \mathbb{V}_5$. The first joint execution of $a^{(1)}$ and $a^{(2)}$ yields $\top^t$, otherwise $\bot^t$; subsequent steps emit the post-frontier verdicts $\top^p$ or $\bot^p$.

Construction for Binary Contract Operators

For binary contract operators of the form $C \text{ op } C'$ with $\text{op} \in \{\wedge, \ ; \ , \blacktriangleright\}$, the monitor for the composite contract is obtained by combining the already constructed monitors $\mathsf{TSMC}(C)$ and $\mathsf{TSMC}(C')$. Each operator has its own monitor construction, defined below for conjunction, sequence, and reparation. We begin with the conjunction case.

Definition 30 (Tight Satisfaction Monitor Construction for Conjunction)**.** *Let C and C′ be contracts in* TACNL *, and let*

$$\mathsf{TSMC}(C) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5) \quad \textit{and} \quad \mathsf{TSMC}(C') = (Q', q_0', \Gamma, \mathbb{V}_5, \delta', \lambda_5').$$

The tight satisfaction monitor construction for the conjunction $C \wedge C'$, written $\mathsf{TSMC}_\wedge(C, C')$, is defined as:

$$\mathsf{TSMC}_\wedge(C, C') = (Q_\wedge, q_0^\wedge, \Gamma, \mathbb{V}_5, \delta_\wedge, \lambda_5^\wedge).$$

- *The state set is the Cartesian product*

$$Q_\wedge = Q \times Q',$$

  with the initial state

$$q_0^\wedge = (q_0, q_0').$$

- *The output function is*

$$\lambda_5^\wedge(x, y) = \lambda_5^{comb}\big(\lambda_5(x),\ \lambda_5'(y)\big),$$

  where λ_5^{comb} is the conjunction-combination table:

$\lambda_5^{comb}(v_1, v_2)$	$?$	$\top^t$	$\bot^t$	$\top^p$	$\bot^p$
$?$	$?$	$?$	$\bot^t$	$?$	$\bot^p$
$\top^t$	$?$	$\top^t$	$\bot^t$	$\top^t$	$\bot^p$
$\bot^t$	$\bot^t$	$\bot^t$	$\bot^t$	$\bot^t$	$\bot^p$
$\top^p$	$?$	$\top^t$	$\bot^t$	$\top^p$	$\bot^p$
$\bot^p$	$\bot^p$	$\bot^p$	$\bot^p$	$\bot^p$	$\bot^p$

- *The transition function is the synchronous product:*

$$\delta_\wedge\big((x, y), A\big) = \big(\delta(x, A),\ \delta'(y, A)\big),$$

  for all $(x, y) \in Q_\wedge$ and $A \in \Gamma$.

Intuition. Each cell from $\lambda_5^{\mathsf{comb}}$ definition specifies the global verdict emitted by the product monitor when the left component is in state x with verdict $\lambda(x)$ and the right component in y with verdict $\lambda'(y)$. The operator is symmetric, commutative, and idempotent. **Partial dominance for $\wedge$. Partial dominance rules for $\wedge$.** The monitor for $C \wedge C'$ checks both components at the same time and decides the global verdict according to the following rules:

- If any component gives $\bot^p$, the result is $\bot^p$:

  $$\forall v \in \mathbb{V}_5 : \quad \bot^p \sqcap v = \bot^p,$$

  That is, once a permanent violation appears, the whole conjunction is permanently violated.

- If any component gives $\bot^t$, and none is permanent, the result is $\bot^t$:

  $$\forall v \in \{?, \top^t, \top^p\} : \quad \bot^t \sqcap v = \bot^t,$$

  That is, a single tight violation makes the conjunction fail tightly.

- Tight and permanent success combine as the weakest success:

  $$\top^t \sqcap \top^t = \top^t, \qquad \top^t \sqcap \top^p = \top^t, \qquad \top^p \sqcap \top^p = \top^p,$$

  The conjunction is only permanently satisfied when both parts are permanent.

- If both sides are undecided or only partly satisfied, the result stays ?:

  $$? \sqcap v = ? \quad \text{for } v \in \{?, \top^t, \top^p\},$$

  The monitor waits until a clear outcome appears.

- The operator is symmetric:

  $$v_1 \sqcap v_2 = v_2 \sqcap v_1,$$

  The order of operands does not matter.

Lemma 10 (Correctness of the conjunctive monitor construction)**.** *Let*

$$\mathsf{TSMC}(C) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5) \quad \textit{and} \quad \mathsf{TSMC}(C') = (Q', q_0', \Gamma, \mathbb{V}_5, \delta', \lambda_5')$$

and let

$$\mathsf{TSMC}(C \wedge C') = (Q_\wedge, q_0^\wedge, \Gamma, \mathbb{V}_5, \delta_\wedge, \lambda_5^\wedge)$$

be the conjunction monitor defined in Definition 30. For every trace π, the output of the conjunction monitor satisfies:

$$\lambda_5^\wedge(\delta_\wedge(q_0^\wedge, \pi)) = [\![\pi \vDash C \wedge C']\!]_5 .$$

Proof sketch. The monitor $\mathsf{TSMC}_\wedge(\mathsf{TSMC}(C), \mathsf{TSMC}(C'))$ runs both component monitors in parallel and computes its output using the conjunction-combination table $\lambda_5^{\mathsf{comb}}$. The correctness follows from the prefix-based tight semantics of $C \wedge C'$.

- Permanent violation in either component produces $\bot^p$ immediately.
- A tight violation in one component produces $\bot^t$ whenever no permanent result is already present.

- Satisfaction requires both components to reach satisfaction states. If one is in $\top^p$ or $\top^t$ and the other is non-violating, the combined verdict matches the corresponding entry in the table.

- If both components remain undecided, the output is ?.

These cases match exactly the clauses for tight satisfaction, tight violation, post-satisfaction, and post-violation for the contract $C \wedge C'$. The monitor, therefore, correctly implements the tight semantics of conjunction. □

Sequential composition is the second binary operator of TACNL . Given two already constructed monitors $\mathsf{TSMC}(C)$ and $\mathsf{TSMC}(C')$, the monitor for the composite contract $C;C'$ must first execute C on the incoming trace and, once C reaches tight satisfaction, must continue execution with C' on the remaining suffix. The construction below reuses the state spaces of both components by redirecting transitions that correspond to the tight success of C into the initial state of C'. This yields a tight prefix monitor that exactly matches the semantics of sequential composition.

Definition 31 (Sequential Monitor Construction)**.** *Let*

$$\mathsf{TSMC}(C) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5) \quad \textit{and} \quad \mathsf{TSMC}(C') = (Q', q_0', \Gamma, \mathbb{V}_5, \delta', \lambda_5')$$

be the tight satisfaction monitors of C and C'. The tight satisfaction monitor for the sequential composition $C;C'$, written $\mathsf{TSMC}_;(C,C')$, is defined as:

$$\mathsf{TSMC}_;(\mathsf{TSMC}(C), \mathsf{TSMC}(C')) = (Q_;, q_0^;, \Gamma, \mathbb{V}_5, \delta_;, \lambda_5^;).$$

- *The state set is*
$$Q_; = (Q \setminus Q^+) \cup Q',$$
where
$$Q^+ = \{x \in Q \mid \lambda_5(x) \in \{\top^t, \top^p\}\},$$
and the initial state is
$$q_0^; = q_0.$$

- *The transition function is*
$$\delta_;(q,A) = \begin{cases} q_0' & \textit{if } q \in Q \textit{ and } \lambda_5(\delta(q,A)) = \top^t, \\ \delta(x,A) & \textit{if } q \in Q \textit{ and } \lambda_5(\delta(q,A)) \notin \{\top^t, \top^p\}, \\ \delta'(x,A) & \textit{if } q \in Q'. \end{cases}$$

- *The output function is*
$$\lambda_5^;(x) = \begin{cases} \lambda_5(x) & \textit{if } x \in Q, \\ \lambda_5'(x) & \textit{if } x \in Q'. \end{cases}$$

Intuition. The construction implements the idea that C must succeed tightly before C' becomes active. All states of C that already correspond to tight satisfaction ($\lambda_5(x) = \top^t$ or $\top^p$) are removed, since execution should never continue inside them. Any transition in C that would have entered such a removed state is redirected to the initial state q'_0 of C', thereby starting the second contract at the exact prefix where C achieves tight success. All other transitions behave exactly as in the original monitors. The resulting machine therefore behaves as C until C succeeds tightly, after which it behaves as C' for the remainder of the trace.

Lemma 11 (Correctness of the sequential monitor construction)**.** *Let* $\mathsf{TSMC}(C)$ *and* $\mathsf{TSMC}(C')$ *be the monitors for C and C', and let* $\mathsf{TSMC}_{;}(\mathsf{TSMC}(C), \mathsf{TSMC}(C')) = (Q_{;}, q^{;}_0, \Gamma, \mathbb{V}_5, \delta_{;}, \lambda^{;}_5)$*. For every trace* π*, the monitor outputs the same verdict according to the sequence semantics:*

$$\lambda^{;}_5\big(\delta_{;}(q^{;}_0, \pi)\big) = [\![\pi \models C\ ;\ C']\!]_5 .$$

Proof. The proof is by induction on the length of the input trace π.

Base case. For $\pi = \varepsilon$, the composite monitor starts in q_0, so $\lambda^{;}_5(\varepsilon) = \lambda_5(q_0)$, which matches the semantics of $C;C'$ on the empty trace.

Inductive step. Assume correctness for all prefixes up to length n and consider the prefix $\pi[n+1]$. The only difference between $\delta_{;}$ and the component transitions is the redirection rule triggered when $\lambda_5(\delta(x,A)) = \top^t$. This redirection starts C' on the remaining suffix of the trace, which matches the semantic clause

$$\exists k:\ \pi_k\ \models_{\top^t}\ C\ \text{ and }\ \pi^{k+1}\ \models_{\top^t}\ C'.$$

All other transitions follow δ or δ' and thus satisfy the inductive hypothesis.

Violation case. If C violates before any tight success, the redirection never occurs, and the global verdict is exactly the violation of C, matching $\pi\ \models_{\bot^t}\ C;C'$.

Satisfaction case. If C reaches tight success at some position k and C' satisfies the suffix π^{k+1}, the transition to q'_0 is activated and the verdict of C' is propagated, matching the semantics of $\models_{\top^t}\ C;C'$.

Conclusion. Every prefix of π is handled by either: (1) normal execution of C or C', or (2) the single redirection step that hands control from C to C'. All verdicts, therefore, coincide exactly with the tight prefix semantics of $C;C'$. $\square$

Definition 32 (Reparation Monitor Construction)**.** *Let*

$$\mathsf{TSMC}(C) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5) \quad \textit{and} \quad \mathsf{TSMC}(C') = (Q', q'_0, \Gamma, \mathbb{V}_5, \delta', \lambda'_5)$$

be the tight satisfaction monitors for C and C'. The monitor for the reparation contract $C \blacktriangleright C'$*, written*

$$\mathsf{TSMC}_{\blacktriangleright}(C, C'),$$

is defined as the tuple

$$\mathsf{TSMC}_{\blacktriangleright}(\mathsf{TSMC}(C), \mathsf{TSMC}(C')) = (Q_{\blacktriangleright}, q^{\blacktriangleright}_0, \Gamma, \mathbb{V}_5, \delta_{\blacktriangleright}, \lambda^{\blacktriangleright}_5).$$

- *The state set is*
$$Q_{\blacktriangleright} = (Q \setminus Q^{-}) \cup Q',$$
where
$$Q^{-} = \{\, q \in Q \mid \lambda_5(q) \in \{\bot^t, \bot^p\} \,\},$$
and the initial state is
$$q_0^{\blacktriangleright} = q_0.$$

- *The transition function is*

$$\delta_{\blacktriangleright}(q,A) = \begin{cases} q'_0 & \text{if } q \in Q \text{ and } \lambda_5(\delta(q,A)) = \bot^t, \\ \delta(q,A) & \text{if } q \in Q \text{ and } \lambda_5(\delta(q,A)) \notin \{\bot^t, \bot^p\}, \\ \delta'(q,A) & \text{if } q \in Q'. \end{cases}$$

- *The output function is*

$$\lambda_5^{\blacktriangleright}(q) = \begin{cases} \lambda_5(q) & \text{if } q \in Q, \\ \lambda'_5(q) & \text{if } q \in Q'. \end{cases}$$

Intuition. The reparation operator activates the secondary contract C' after the primary contract C reaches a tight failure. The construction mirrors the sequential case, except that the redirection applies to transitions leading to a tight violation of C rather than to those leading to tight satisfaction.

All states of C whose verdicts are already violating ($\lambda_5(q) \in \{\bot^t, \bot^p\}$) are removed. Every transition in C that would enter such a state is redirected to q'_0, the initial state of C'. This starts C' exactly at the first prefix where C tightly fails.

As long as no violation occurs, the monitor behaves exactly as C. Once a tight failure is detected, the remaining input is processed by C'.

Lemma 12 (Correctness of the reparation monitor construction)**.** *Let* $\mathsf{TSMC}(C)$ *and* $\mathsf{TSMC}(C')$ *be the monitors for C and C', and let*

$$\mathsf{TSMC}_{\blacktriangleright}(\mathsf{TSMC}(C), \mathsf{TSMC}(C')) = (Q_{\blacktriangleright}, q_0^{\blacktriangleright}, \Gamma, \mathbb{V}_5, \delta_{\blacktriangleright}, \lambda_5^{\blacktriangleright})$$

be the monitor constructed in Definition 32. Then, for every trace π, the monitor outputs the right verdict as specified by the tight semantics:

$$\lambda_5^{\blacktriangleright}(\delta_{\blacktriangleright}(q_0^{\blacktriangleright}, \pi)) = [\![\pi \vDash C \blacktriangleright C']\!]_5.$$

Proof. The argument parallels the proof for sequential composition.

Base case. For $\pi = \varepsilon$, the monitor begins in $q_0^{\blacktriangleright} = q_0$. Thus

$$\lambda_5^{\blacktriangleright}(\delta_{\blacktriangleright}(q_0^{\blacktriangleright}, \varepsilon)) = \lambda_5(q_0),$$

which matches the tight semantics of $C \blacktriangleright C'$ on the empty trace.

Inductive step. Assume correctness for prefixes up to length n and consider the next letter A. If $\lambda_5(\delta(x,A)) = \bot^t$, then $\delta_{\blacktriangleright}(x,A) = q'_0$, so C' takes over on the remaining suffix. This matches the semantic clause

$$\exists k : \pi_k \models_{\bot^t} C \quad \text{and} \quad \pi^{k+1} \models_{\top^t} C'.$$

If no tight violation occurs, $\delta_{\blacktriangleright}$ behaves exactly as δ, so the inductive hypothesis applies.

Violation case. If C reaches a tight failure and C' later fails on the suffix, the monitor outputs $\bot^p$, exactly as prescribed by the semantics.

Satisfaction case. If C tightly fails and C' succeeds on the suffix, the monitor outputs $\top^t$ or $\top^p$, depending on whether the success occurs tightly or post-satisfaction.

Conclusion. Every step of $\delta_{\blacktriangleright}$ corresponds either to execution of C, execution of C', or the single switch determined by tight failure of C. Thus, $\lambda_5^{\blacktriangleright}(\delta_{\blacktriangleright}(q_0^{\blacktriangleright}, \pi))$ matches exactly the tight semantics of $C \blacktriangleright C'$. □

Remark 2 (Relation to Control Phases and Prior Work)**.** *The structural constructions of Definitions 31 and 32 can be seen as the static counterpart of the* control-phase *view used in runtime-verification frameworks. Instead of introducing an explicit control variable $m \in \{\mathbf{L}, \mathbf{R}\}$, with $\mathbf{L}$ for left-hand side and $\mathbf{R}$ for the right, to determine which component is active, our transformation achieves the same effect directly on the transition graph deleting the terminal states of the left monitor and redirecting the transitions that reach a decisive verdict ($\top^t$ for sequence, $\bot^t$ for reparation) to the initial state of the right monitor. This compile-time construction encodes the same operational behavior as a mode-augmented monitor that switches from $\mathbf{L}$ to $\mathbf{R}$ when the switching condition is met.*

*This idea parallels the phase-based runtime semantics proposed in the runtime verification literature [8, 4], where control modes are used to synchronize sub-monitors for sequential patterns such as "*after C succeeds, check C'*". In contrast, the reparation composition corresponds to the "*compensatory phase*" discussed in [9], where a secondary clause is activated after the primary obligation fails. Hence, the constructions in Definitions 31 and 32 realize at the automaton level the same phase shifts ($\mathbf{L} \to \mathbf{R}$ after tight success or tight failure) that those frameworks handle explicitly through control variables.*

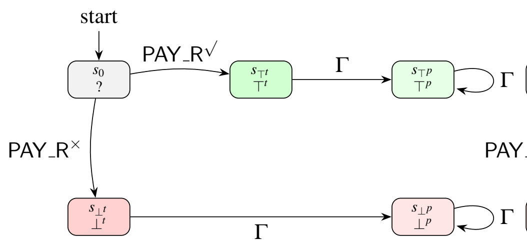
Example 23 (5-Output Moore Monitors for C_3 and $C_2 \wedge C_3$)**.** *The reparation contract $C_3 = \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ combines two obligations: the primary duty $\mathbf{O}_1(\mathsf{PAY\_R})$ to pay rent, and the secondary reparation $\mathbf{O}_1(\mathsf{PAY\_F})$ to pay a late fee if the first is violated. In addition, $C_2 = \mathbf{P}_1(\mathsf{OCC})$ grants the tenant (agent 2) permission to occupy the property. Below, we show the literal monitors, their reparation composition, and the conjunction $C_2 \wedge C_3$, with verdict outputs $\mathbb{V}_5 = \{?, \top^t, \bot^t, \top^p, \bot^p\}$.*

We construct $C_2 \wedge C_3$ by the synchronous product of the 5-output monitors for $C_2 = \mathbf{P}_2(\mathsf{OCC})$ and $C_3 = \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$, then keep only reachable states and minimize construction 10. The Letter classes used on edges correspond to shorthand *for literal satisfaction or violation conditions:*

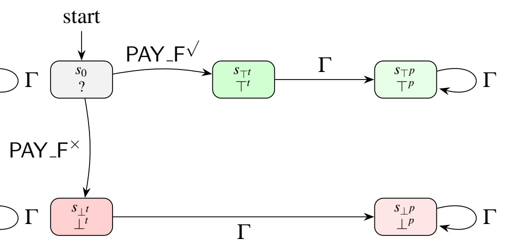
$$\mathsf{PAY\_R}^{\checkmark} := \{A \in \Gamma \mid \{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\} \subseteq A\}, \qquad \mathsf{PAY\_R}^{\times} := \Gamma \setminus \mathsf{PAY\_R}^{\checkmark},$$
$$\mathsf{PAY\_F}^{\checkmark} := \{A \in \Gamma \mid \{\mathsf{PAY\_F}^{(1)}, \mathsf{PAY\_F}^{(2)}\} \subseteq A\}, \qquad \mathsf{PAY\_F}^{\times} := \Gamma \setminus \mathsf{PAY\_F}^{\checkmark},$$
$$\mathsf{OCC}^{\checkmark} := \{A \in \Gamma \mid \mathsf{OCC}^{(2)} \in A \Rightarrow \mathsf{OCC}^{(1)} \in A\}, \qquad \mathsf{OCC}^{\times} := \Gamma \setminus \mathsf{OCC}^{\checkmark}.$$

Outputs follow the conjunction rule Λ: $\bot^t$ if any conjunct tightly rejects, $\bot^p$ if any conjunct is post-reject, $\top^t$ when one conjunct hits $\top^t$ while the other is already at/past acceptance ($\top^t$ or $\top^p$), $\top^p$ if both are post-accept, and $?$ otherwise.

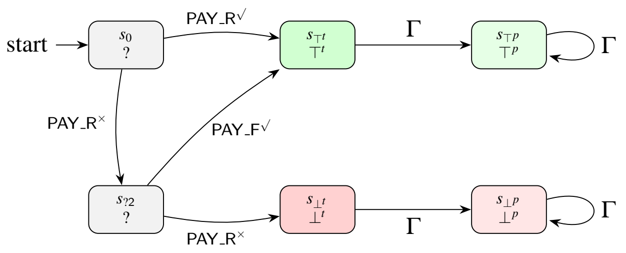
Reading Fig. 10d with the letter classes defined above: from s_0 (pre) there are three behaviors: (i) If $\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\checkmark}$ holds in the current letter, both conjuncts succeed (perm is respected, and the primary obligation is met), so the product emits $\top^t$ and moves to post-accept $\top^p$. (ii) If $\mathsf{OCC}^{\times}$ holds, the permission conjunct fails tightly, so the product emits $\bot^t$ and then $\bot^p$ forever. (iii) If $\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\times}$ holds, the reparation branch of C_3 is activated, and we move to the waiting state s_1 (still $?$). From s_1, $\mathsf{PAY\_F}^{\checkmark}$ discharges the reparation and triggers $\top^t$; otherwise $\mathsf{PAY\_F}^{\times}$ yields $\bot^t$. After $\top^t$ (accepting frontier), all continuations are in $\top^p$; after $\bot^t$ (reject frontier) all continuations are in $\bot^p$. Thus, the decisive index for the conjunction is the latter of the two successes when both succeed, or the earlier tight failure when any conjunct fails, exactly as the figure shows.



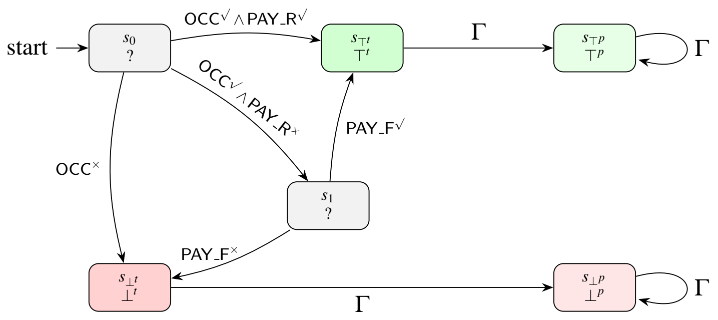
(a) $\mathbf{O}_1(\mathsf{PAY\_R})$. Satisfaction occurs when both agents perform PAY_R.



(b) $\mathbf{O}_1(\mathsf{PAY\_F})$. Satisfaction occurs when both agents perform PAY_F.



(c) Reparation composition $\mathsf{TSMC}_{\blacktriangleright}(\mathbf{O}_1(\mathsf{PAY\_R}), \mathbf{O}_1(\mathsf{PAY\_F}))$. The monitor activates PAY_F after tight failure of PAY_R.



(d) Conjunctive composition $\mathsf{TSMC}_{\wedge}(C_2, C_3)$, where $C_2 = \mathbf{P}_1(\mathsf{OCC})$.

Figure 10: Literal and composite 5-output Moore monitors for $C_3 = \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$. (a) and (b) monitors for Literal composing C_3 side by side; (c) Reparation composition C_3; (d) Conjunctive composition $C_2 \wedge C_3$, where $C_2 = \mathbf{P}_1(\mathsf{OCC})$ represents the tenant's power to occupy the property. $\mathsf{OCC}^{\checkmark} := \{A \mid \{\mathsf{OCC}^1\} \not\subseteq A$ or $\{\mathsf{OCC}^1, \mathsf{OCC}^1\} \subseteq A\}$. $\mathsf{PAY\_R}^{\checkmark} := \{A \mid \mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\} \subseteq A$. The case $\mathsf{PAY\_F}^{\checkmark}$ is similarly defined as for $\mathsf{PAY\_R}^{\checkmark}$.

Construction for Binary Regular Expression-Contracts

Triggered contracts activate their body C once the triggering pattern *re* reaches its first tight match. Before that point, the monitor behaves exactly as the regular-expression monitor for *re* and emits only ?. Once the trigger fires, the monitor switches permanently to the contract monitor $\mathsf{TSMC}(C)$. If the pattern becomes impossible before it fires, the contract is vacuously satisfied. The construction follows the same blueprint as sequence and reparation, but applied to the decisive states of the regular expression monitor.

Definition 33 (Triggered Monitor Construction)**.**

$$\textit{Let } \mathsf{TSMC}(\mathsf{re}) = (Q_r, r_0, \Gamma, \mathbb{V}_5, \delta_r, \lambda_5^r) \quad \textit{and} \quad \mathsf{TSMC}(C) = (Q_c, c_0, \Gamma, \mathbb{V}_5, \delta_c, \lambda_5^c)$$

be five-valued tight satisfaction monitors for the regular expression re *and the contract C. The triggered monitor for* $\langle\langle re\rangle\rangle C$ *is the machine*

$$\mathsf{TSMC}_{trig}(\mathsf{TSMC}_{re}(\mathsf{re}), \mathsf{TSMC}(C)) := (Q_{trig}, q_0^{trig}, \Gamma, \mathbb{V}_5, \delta_{trig}, \lambda_5^{trig}).$$

- *The state space is the states from the contracts unified with the reduced regular expression monitor states:*

$$Q_{trig} := Q_r^{open} \cup Q_c,$$

where the reduced regular expressions states are:

$$Q_r^{open} := \{\, q \in Q_r \mid \lambda_5^r(q) \in \{?, \top^t, \top^p\} \,\}.$$

The initial state is

$$q_0^{trig} := r_0.$$

- *The transition function is defined in two parts:*

1. ***Guard-active region*** *(*$q \in Q_r^{\boldsymbol{open}}$*). Let* $q' = \delta_r(q, A)$.

$$\delta_{trig}(q, A) = \begin{cases} c_0 & \textit{if } q' \in Q_r^{\top^t} \quad \textit{(first tight match of re)}, \\ q^t & \textit{if } q' \in Q_r^{\perp^t} \quad \textit{(trigger impossible)}, \\ q' & \textit{if } q' \in Q_r^{?} \quad \textit{(guard still open)}. \end{cases}$$

Here $q^t \in Q_c$ *is any state with* $\lambda_5^c(q^t) = \top^t$ *and* $q^p \in Q_c$ *is the analogous* $\top^p$*-state, reused from the* TSMC*(C) monitor construction.*

2. ***Contract-active region*** *(*$y \in Q_c$*)*

$$\delta_{trig}(y, A) := \delta_c(y, A).$$

- *The output function is*

$$\lambda_5^{trig}(q) = \begin{cases} ? & \textit{if } q \in Q_r^{open}, \\ \lambda_5^c(q) & \textit{if } q \in Q_c. \end{cases}$$

Intuition. The trigger monitor is obtained with the same redirection recipe used for sequence, but applied to the trigger. Keep only states that are still open in the trigger (outputs in $\{?, \top^t, \top^p\}$). Remove its decisive states. Redirect every transition that would be the first tight match of the triggering regular expression (the step that enters $\top^t$) to the initial state of the contract monitor C. From that point on, the global output is exactly the output of C on the suffix. Redirect every transition that would make the trigger impossible (enter $\bot^t$ or $\bot^p$) to a vacuous-success sink for the overall contact that emits $\top^t$. While the guard remains open, only the guard component advances and the product emits ?, so no premature verdict appears. This realizes the prefix clauses: success either because the trigger never becomes true, which we refer to as vacuous satisfaction, or because it fires at the earliest index and the suffix satisfies C; violation only if the guard fires and the suffix violates C.

Lemma 13 (Correctness of the triggered monitor construction)**.**

$$\textit{Let } \mathsf{TSMC}_{trig}(\mathsf{TSMC}_{re}(\mathsf{re}), \mathsf{TSMC}(C)) = (Q_{trig}, \Gamma, \delta_{trig}, q_0^{trig}, \lambda_5^{trig})$$

be the monitor constructed in Definition 33 for $\langle\langle re \rangle\rangle C$. Then, for every trace π,

$$\lambda_5^{trig}\big(\delta_{trig}(q_0^{trig}, \pi)\big) \; = \; [\![\pi \vDash \langle\langle re \rangle\rangle C]\!]_5\,.$$

Proof sketch. The proof follows the same pattern as for sequence and reparation, by induction on the length of π.

Base case. For $\pi = \varepsilon$ the monitor is in $q_0^{\text{trig}} = r_0$, the initial state of the guard. The value

$$\lambda_5^{\text{trig}}\big(\delta_{\text{trig}}(q_0^{\text{trig}}, \varepsilon)\big) = \lambda_5^r(r_0) = ?$$

coincides with the tight semantics of trig$[re]C$ on the empty trace: no trigger has fired, and no violation has occurred.

Inductive step. Assume the invariant holds for all prefixes of length n. Consider a prefix of length $n+1$ and its last letter A.

There are two regions:

- *Guard-active region ($x \in Q_r^{open}$).* By construction, $\delta_{\text{trig}}(x, A)$ is:
  - c_0, if $\delta_r(x, A) \in Q_r^{\top^t}$. This is exactly the case where *re* reaches tight success for the first time. The next state is the initial state of $\mathsf{TSMC}(C)$, so subsequent behavior matches the semantics of C on the suffix. This realizes the clause "trigger fires at the earliest index and the suffix must satisfy C".
  - a vacuous-success state (or the chosen $\top^t$–$\top^p$ pair) if $\delta_r(x, A) \in Q_r^{\bot^t}$, that is, the pattern becomes impossible. This matches the case where the trigger never fires and $\langle\langle re \rangle\rangle C$ holds vacuously.
  - $\delta_r(x, A)$ if $\delta_r(x, A) \in Q_r^?$, in which case the guard remains open, and the global verdict stays ?. This matches the semantic clause that no decisive information is available as long as neither a match nor an impossibility has been detected.

The inductive hypothesis on the guard monitor ensures that the moment of redirection coincides with the earliest decisive prefix of *re*.

- *Contract-active region ($y \in Q_c$).* Once the monitor has been redirected into c_0, all transitions are given by δ_c, and outputs by λ_5^c. The induction hypothesis for $\mathsf{TSMC}(C)$ gives

$$\lambda_5^{\text{trig}}\big(\delta_{\text{trig}}(q_0^{\text{trig}},\pi)\big) = \lambda_5^c\big(\delta_c(c_0,\pi')\big) = [\![\pi' \vDash C]\!]_5,$$

where π' is the suffix after the trigger point. This matches the semantics of $\text{trig}[re]C$ on all traces where the trigger has fired.

Violation and satisfaction cases. If the trigger fires at some earliest index k and the suffix π^{k+1} violates C tightly or post, the monitor is in the contract-active region and outputs the corresponding $\bot^t$ or $\bot^p$, which is exactly $[\![\pi \vDash \text{trig}[re]C]\!]_5$ in this case. If the trigger never fires and the guard becomes impossible, the monitor outputs $\top^t$ and then $\top^p$, which matches vacuous satisfaction. In all other cases the output remains ?, as the semantics of $\text{trig}[re]C$ leaves the status undecided.

Conclusion. At each prefix, the monitor either simulates the guard with the correct decisive redirection points or simulates C on the correct suffix. Hence for every trace π the monitor verdict $\lambda_5^{\text{trig}}(\delta_{\text{trig}}(q_0^{\text{trig}},\pi))$ coincides with the five-valued tight semantics of $\text{trig}[re]C$. □

Definition 34 (Guarded Monitor Construction)**.**

$$\textit{Let } \mathsf{TSMC}(\mathsf{re}) = (Q_r, r_0, \Gamma, \mathbb{V}_5, \delta_r, \lambda_5^r) \quad \textit{and} \quad \mathsf{TSMC}(C) = (Q_c, c_0, \Gamma, \mathbb{V}_5, \delta_c, \lambda_5^c).$$

The guarded contract monitor for $\lceil re \rceil C$ is the synchronous product

$$\mathsf{TSMC}_{guard}(\mathsf{TSMC}_{re}(\mathsf{re}), \mathsf{TSMC}(C)) := (Q_r \times Q_c,\ (r_0,c_0),\ \Gamma,\ \mathbb{V}_5,\ \delta_{guard},\ \lambda_5^{guard}),$$

with

$$\delta_{guard}((x,y),A) := (\delta_r(x,A), \delta_c(y,A)).$$

$$\lambda_5^{guard}(x,y) = \begin{cases} \bot^t & \textit{if } \lambda_5^r(x) \in \{?, \top^t, \top^p\} \textit{ and } \lambda_5^c(y) = \bot^t, \\ \bot^p & \textit{if } \lambda_5^r(x) \in \{?, \top^t, \top^p\} \textit{ and } \lambda_5^c(y) = \bot^p, \\ \top^t & \textit{if } \lambda_5^r(x) = \bot^t \textit{ and } \lambda_5^c(y) \in \{?, \top^t, \top^p\}, \\ \top^p & \textit{if } \lambda_5^r(x) = \bot^p \textit{ and } \lambda_5^c(y) \in \{?, \top^t, \top^p\}, \\ \top^t & \textit{if } \lambda_5^r(x) \in \{\top^t, \top^p\} \textit{ and } \lambda_5^c(y) = \top^t, \\ \top^p & \textit{if } \lambda_5^r(x) \in \{\top^t, \top^p\} \textit{ and } \lambda_5^c(y) = \top^p, \\ ? & \textit{otherwise.} \end{cases}$$

Intuition. The guard reads both monitors in lockstep and enforces:

- *Open guard.* While the guard is open $\big(\lambda_r \in \{?, \top^t, \top^p\}\big)$, i.e. exactly when $\pi \models_{Cl} re$, any tight or post failure of C becomes the global failure:

$$\pi \models_{\bot^t} \lceil re \rceil C \iff (\pi \models_{Cl} re) \wedge (\pi \models_{\bot^t} C),$$
$$\pi \models_{\bot^p} \lceil re \rceil C \iff (\pi \models_{Cl} re) \wedge (\pi \models_{\bot^p} C).$$

- *Guard impossible.* If the guard becomes impossible $\big(\lambda_r \in \{\bot^t, \bot^p\}\big)$, we accept provided C has not failed:

$$\pi \models_{\top^t} \lceil re \rceil C \iff (\pi \models_{\bot^t} re) \wedge (\pi \models_{Cl} C),$$
$$\pi \models_{\top^p} \lceil re \rceil C \iff (\pi \models_{\bot^t} re) \wedge (\pi \models_{\top^p} C).$$

- *Guard fired/closed.* When the guard has fired/closed $\big(\lambda_r \in \{\top^t, \top^p\}\big)$, we require C to (tight/post) succeed:

$$\pi \models_{\top^t} \lceil re \rceil C \iff (\pi \models_{Cl} re) \wedge (\pi \models_{\top^t} C),$$
$$\pi \models_{\top^p} \lceil re \rceil C \iff (\pi \models_{Cl} re) \wedge (\pi \models_{\top^p} C).$$

The resulting case table is symmetric and total, and it collapses to the expected two-valued clauses once tight/post outcomes are merged into satisfied vs. violated.

Lemma 14 (Correctness of the guarded monitor construction)**.**

Let $\mathsf{TSMC}_{guard}(\mathsf{TSMC}_{re}(\mathsf{re}), \mathsf{TSMC}(C)) = (Q_{guard}, q_0^{guard}, \Gamma, \mathbb{V}_5, \delta_{guard}, \lambda_5^{guard})$

be the monitor constructed in Definition 34 for guard[re]C. Then, for every finite trace π,

$$\lambda_5^{guard}\big(\delta_{guard}(q_0^{guard}, \pi)\big) \;=\; [\![\pi \models \lceil re \rceil C]\!]_5\,.$$

Proof sketch. The argument proceeds by induction on the length of π. The guarded monitor is a synchronous product of $\mathsf{TSMC}(re)$ and $\mathsf{TSMC}(C)$, with the output governed by the case distinction in Definition 34. Each region of the case table matches exactly one of the semantic clauses for guard$[re]C$.

Base case. For $\pi = \varepsilon$ we have

$$\lambda_5^{\text{guard}}\big(\delta_{\text{guard}}(q_0^{\text{guard}}, \varepsilon)\big) = \lambda_5^{\text{guard}}(r_0, c_0),$$

which yields ? in agreement with $[\![\varepsilon \models \text{guard}[re]C]\!]_5$.

Inductive step. Assume correctness for all prefixes of length n. Consider $\pi[n{+}1]$ with last letter A and let

$$(x', y') := \delta_{\text{guard}}((x, y), A) = (\delta_r(x, A), \delta_c(y, A)).$$

There are three semantic regions, corresponding to the three guard statuses.

- **Guard open** $\lambda_5^r(x) \in \{?, \top^t, \top^p\}$. This means π still possibly satisfies the guard. The guarded semantics requires that any tight or post failure of C becomes the global failure. The monitor table assigns $\bot^t$ or $\bot^p$ precisely in these cases, and ? otherwise, matching

$$[\![\pi \models \text{guard}[re]C]\!]_5 = [\![\pi \models C]\!]_5 \quad \text{as long as } re \text{ is still open.}$$

- **Guard impossible** $\lambda_5^r(x) \in \{\bot^t, \bot^p\}$. This corresponds exactly to $\pi \models_{\bot^t} re$. The guarded semantics declares vacuous acceptance provided that C has not already failed. The output table assigns $\top^t$ or $\top^p$ if C has not failed, and propagates $\bot^t$ or $\bot^p$ if it has. This matches the semantic requirements for vacuous satisfaction.

- **Guard closed (triggered or concluded)** $\lambda_5^r(x) \in \{\top^t, \top^p\}$. In this region, the guard has fired or completed successfully, and the semantics require C to satisfy or to fail. The output table combines the post-accept and tight-accept verdicts of C with those of re exactly as demanded by the five-valued semantics:

$$[\![\pi \models \text{guard}[re]C]\!]_5 = [\![\pi \models C]\!]_5 \quad \text{once } re \text{ has closed.}$$

Conclusion. At each prefix of π, the guarded monitor outputs exactly the verdict prescribed by the five-valued semantics of guard$[re]C$. Thus

$$\lambda_5^{\text{guard}}\big(\delta_{\text{guard}}(q_0^{\text{guard}}, \pi)\big) = [\![\pi \models \text{guard}[re]C]\!]_5$$

for all traces π. □

Construction for repetition contracts

Repetition contracts describe behaviors that must be satisfied several times in sequence. The monitor construction follows the intuition that each repetition runs an independent copy of the monitor for C, with the next copy becoming active exactly when the previous one reaches tight success. For finite repetition C^n, this results in n chained monitors. For unbounded repetition $\mathbf{Rep}(C)$, we obtain an infinite cycle without ever reporting tight success.

The following constructions make these ideas explicit.

Definition 35 (Tight monitor construction for finite repetition frep(n,C))**.**

$$\textit{Let } \mathsf{TSMC}(C) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5)$$

be the five-valued tight satisfaction monitor for C. For n $\in \mathbb{N}^*$*, the tight monitor for the finite repetition contract frep*(n,C) *is defined as*

$$\mathsf{TSMC}_{\textit{frep}}(n,C) := (Q_{\textit{frep}}, q_0^{\textit{frep}}, \Gamma, \mathbb{V}_5, \delta_{\textit{frep}}, \lambda_5^{\textit{frep}}).$$

Disjoint copies. *For each i* $\in \{1,\ldots,n\}$*, create a disjoint copy of the base monitor:*

$$Q^{(i)} = \{q^{(i)} \mid q \in Q\}, \qquad \delta^{(i)}(q^{(i)}, A) = (\delta(q,A))^{(i)}, \qquad \lambda_5^{(i)}(q^{(i)}) = \lambda_5(q).$$

Write $q_0^{(i)}$ for the copy of q_0.

State space and start state.

$$Q_{frep} := \Big(\bigcup_{i=1}^{n} Q^{(i)}\Big) \cup \{q_{\top^p}, q_{\bot^p}\}, \qquad q_0^{frep} := q_0^{(1)},$$

where $q_{\top^p}$ and $q_{\bot^p}$ are fresh post-success and post-failure sinks.

Transition function. *For $1 \le i \le n-1$:*

$$\delta_{frep}(q^{(i)}, A) = \begin{cases} q_0^{(i+1)} & \text{if } \lambda_5^{(i)}(\delta^{(i)}(q^{(i)}, A)) = \top^t, \\ \delta^{(i)}(q^{(i)}, A) & \text{if } \lambda_5^{(i)}(\delta^{(i)}(q^{(i)}, A)) \notin \{\top^t, \top^p\}. \end{cases}$$

For the last copy $i = n$:

$$\delta_{frep}(q^{(n)}, A) = \begin{cases} q_{\top^p} & \text{if } \lambda_5^{(n)}(\delta^{(n)}(q^{(n)}, A)) = \top^t, \\ \delta^{(n)}(q^{(n)}, A) & \text{if } \lambda_5^{(n)}(\delta^{(n)}(q^{(n)}, A)) \notin \{\top^t, \top^p\}. \end{cases}$$

Sink states absorb:

$$\delta_{frep}(q_{\top^p}, A) = q_{\top^p}, \qquad \delta_{frep}(q_{\bot^p}, A) = q_{\bot^p}.$$

Output function. *For $q^{(i)} \in Q^{(i)}$:*

$$\lambda_5^{frep}(q^{(i)}) = \begin{cases} \bot^t & \text{if } \lambda_5^{(i)}(q^{(i)}) = \bot^t, \\ ? & \text{if } \lambda_5^{(i)}(q^{(i)}) \in \{?, \top^t, \top^p\}. \end{cases}$$

For sink states:

$$\lambda_5^{frep}(q_{\top^p}) = \top^p, \qquad \lambda_5^{frep}(q_{\bot^p}) = \bot^p.$$

This completes the construction of the five-valued monitor for $frep(n, C)$.

Lemma 15 (Correctness of the finite repetition monitor)**.** *For every finite trace π, contract C, and $\mathsf{TSMC}_{frep}(n, C) := (Q_{frep}, q_0^{frep}, \Gamma, \mathbb{V}_5, \delta_{frep}, \lambda_5^{frep})$, the following holds*

$$\lambda_5^{frep}\big(\delta_{frep}(q_0^{frep}, \pi)\big) = [\![\pi \vDash C^n]\!]_5 .$$

In the next sections, we use the definitions of the different languages in the semantics of TACNL , along with automata constructions and transformations, to enable automatic detection of violations and the attribution of blame for synchronous interactions over contracts in TACNL .

Definition 36 (Tight monitor construction for unbounded repetition **Rep**(C))**.** *Let*

$$\mathsf{TSMC}(C) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5)$$

be the tight satisfaction monitor for C. The monitor for the unbounded repetition $\mathbf{Rep}(C)$ *is defined as*

$$\mathsf{TSMC}_{Rep} := (Q_\omega, q_0, \Gamma, \mathbb{V}_5, \delta_\omega, \lambda_5^\omega)$$

where the construction removes all success states of C and redirects tight success back to the initial state.

State space.

$$Q_\omega := Q \setminus Q_\top, \qquad Q_\top := \{\, q \in Q \mid \lambda_5(q) \in \{\top^t, \top^p\} \,\}.$$

Transition function. *For every $q \in Q_\omega$ and $A \in \Gamma$:*

$$\delta_\omega(q,A) = \begin{cases} q_0 & \textit{if } \lambda_5(\delta(q,A)) = \top^t \quad (\textit{restart next iteration after tight success}), \\ \delta(q,A) & \textit{otherwise.} \end{cases}$$

Output function. *The monitor never declares satisfaction:*

$$\lambda_5^\omega(q) = \begin{cases} \bot^t & \textit{if } \lambda_5(q) = \bot^t, \\ ? & \textit{if } \lambda_5(q) \in \{?, \top^t, \top^p\}. \end{cases}$$

This yields the tight five-valued monitor for $\mathbf{Rep}(C)$.

Lemma 16 (Correctness of the unbounded repetition monitor)**.** *For every finite trace π, and* $\mathsf{TSMC}_{\mathbf{Rep}()}(C) := (Q_\omega, q_0, \Gamma, \mathbb{V}_5, \delta_\omega, \lambda_5^\omega)$ *, the following holds*

$$\lambda_5^\omega\big(\delta_\omega(q_0, \pi)\big) = [\![\pi \vDash \mathbf{Rep}(C)]\!]_5 .$$

The monitor never emits $\top^t$ nor $\top^p$, because tight satisfaction of $\mathbf{Rep}(C)$ is false. It emits $\bot^t$ (and then permanently $\bot^p$) exactly when some iteration of C violates tightly, i.e.

$$\exists n \in \mathbb{N} : \ \pi \models_{\bot^t} C^n.$$

Otherwise, it remains in the undecided verdict ?.

Example 24 (Open ended contract monitor construction)**.** *W*

Example 25 (Open ended contract monitor construction)**.** *We illustrate the constructions for the guarded open-ended contract* $\lceil re \rceil \mathbf{Rep}(C_3)$ *where* $re = *^+; \{\mathsf{Notif\_T}^{(1)}\}$ *and* $C_3 = \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$. *The construction proceeds in three steps: the unbounded-repetition monitor for C_3, the regular-expression monitor for re, and finally the guarded product. The resulting automata are shown in Figure 11.*

(a) $\mathsf{TSMC}(\mathbf{Rep}(C_3))$ ***(Subfigure 11a).*** *The monitor contains the states $q_0/?$, $q_w/?$, $q_{\bot^t}/\bot^t$, and $q_{\bot^p}/\bot^p$. The meaning of the transitions is as follows.*

A joint payment $\mathsf{PAY\_R}^{\surd}$ *keeps the monitor at q_0. A missed payment* $\mathsf{PAY\_R}^{\times}$ *moves to the waiting state q_w. From q_w, a successful late fee* $\mathsf{PAY\_F}^{\surd}$ *restarts the cycle* *by returning to q_0. A failed repair $\mathsf{PAY\_F}^{\times}$ produces a tight violation and moves to $q_{\perp^t}$, which then steps on any letter to the permanent sink $q_{\perp^p}$. The sink loops on all letters.*

This matches the construction of Definition 36: tight success restarts a new cycle, and tight failure leads to a permanent violation.

(b) $\mathsf{TSMC}_{re}(re)$ ***for*** *$re = *^+;\{\mathsf{Notif\_T}^{(1)}\}$* ***(Subfigure 11b).*** *The monitor has states $s_0/?$, $s_1/?$, $s^t/\top^t$, and $s^+/\top^p$.*

The regular-expression part reads arbitrary letters: $s_0 \xrightarrow{*} s_1$. While no termination notice is received, the machine remains in s_1 via $\overline{T} = \Gamma \setminus T$. A letter in $T = \{A \mid \mathsf{Notif\_T}^{(1)} \in A\}$ produces a tight match and moves to s^t. Any continuation moves to the post-acceptance state s^+, which loops on all letters.

(c) Guarded product $\mathsf{TSMC}_{\boldsymbol{guard}}(\mathsf{TSMC}_{re}(re), \mathsf{TSMC}(\mathbf{Rep}(C_3)))$ ***(Subfigure 11c).*** *The product is organised by verdict class: violating states on the left, undecided states in the centre, and accepting states on the right.*

Undecided region (centre). *The reachable combinations while the guard is open are $s_0 \times q_0$, $s_1 \times q_w$, and $s_1 \times q_0$. Their transitions follow the product rule $(x,y) \xrightarrow{A} (\delta_r(x,A), \delta_c(y,A))$. Examples include:*

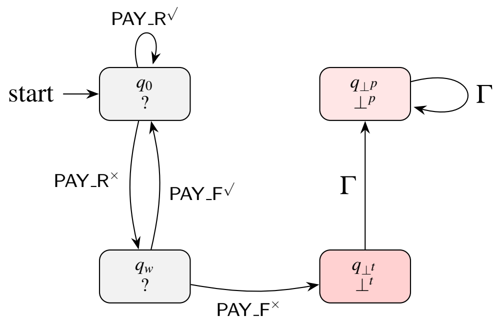
$$\begin{aligned}
s_0 \times q_0 &\xrightarrow{\mathsf{PAY\_R}^{\times}} s_1 \times q_w,\\
s_1 \times q_w &\xrightarrow{\mathsf{PAY\_F}^{\times}} (s_0, s_1) \times q_{\perp^t},\\
s_1 \times q_w &\xrightarrow{\mathsf{PAY\_F}^{\checkmark} \wedge \overline{T}} s_1 \times q_0,\\
s_1 \times q_0 &\xrightarrow{\mathsf{PAY\_R}^{\times} \wedge \overline{T}} s_1 \times q_w,\\
s_0 \times q_0 &\xrightarrow{\mathsf{PAY\_R}^{\checkmark}} s_1 \times q_0.
\end{aligned}$$

Violation region (left). *Once the contract component reaches $q_{\perp^t}$, the guard is still open so the product outputs $\perp^t$ and moves on any letter to $S \times q_{\perp^p}$, which loops on every letter.*

Acceptance region (right). *When the guard fires on a letter in T, the product moves to $s^t \times (q_0, q_w)$ with verdict $\top^t$ provided the repetition contract has not violated. From there, any continuation leads to the post-acceptance state $s^+ \times Q$ that loops on all letters and emits $\top^p$.*

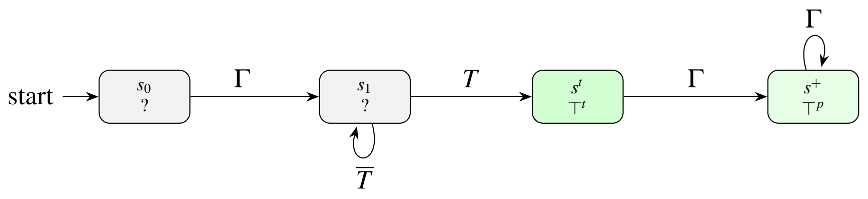
This behavior follows directly from Definition 34: while $\lambda_r \in \{?, \top^t, \top^p\}$ the guard is open, so any tight or post violation of $\mathbf{Rep}(C_3)$ becomes a violation of the guarded contract. Once the guard becomes true on T, the contract must satisfy $\mathbf{Rep}(C_3)$ from that point on, for the global verdict to be tight or post acceptance.

The guarded open-ended contract illustrates how the automaton constructions combine. The unbounded repetition of C_3 enforces an indefinite sequence of payments with repair, and the regular expression *re* recognizes the first occurrence of a termination notice. The guarded product synchronizes both behaviors: while the guard is open, all



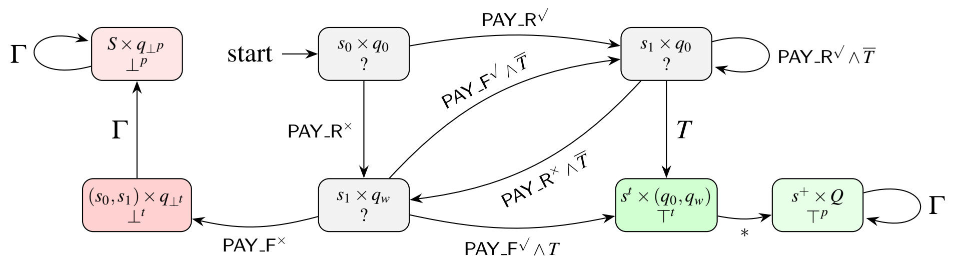
start
q_0 ?
$\mathsf{PAY\_R}^{\surd}$
$\mathsf{PAY\_R}^{\times}$
$\mathsf{PAY\_F}^{\surd}$
q_w ?
$\mathsf{PAY\_F}^{\times}$
$q_{\perp^t}$ $\perp^t$
Γ
$q_{\perp^p}$ $\perp^p$
Γ

(a) Monitor for $\mathbf{Rep}(C_3)$.



$$T := \{A \in \Gamma \mid \mathsf{Notif\_T}^{(1)} \in A\},\ \overline{T} := \Gamma \backslash T,$$

(b) Monitor for the regular expression $\Gamma^+ \cdot\ \{\mathit{terminate}^{(1)}\}$.



Γ
$S \times q_{\perp^p}$ $\perp^p$
start
$s_0 \times q_0$?
$\mathsf{PAY\_R}^{\surd}$
$s_1 \times q_0$?
$\mathsf{PAY\_R}^{\surd} \wedge \overline{T}$
$\mathsf{PAY\_F}^{\surd} \wedge \overline{T}$
$\mathsf{PAY\_R}^{\times}$
Γ
T
$\mathsf{PAY\_R}^{\times} \wedge \overline{T}$
$(s_0, s_1) \times q_{\perp^t}$ $\perp^t$
$s_1 \times q_w$?
$s^t \times (q_0, q_w)$ $\top^t$
$s^+ \times Q$ $\top^p$
Γ
$\mathsf{PAY\_F}^{\times}$
$\mathsf{PAY\_F}^{\surd} \wedge T$
$*$

(c) Tight satisfaction monitor construction for $\lceil \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)} \rceil \mathbf{Rep}(C_3)$.

Figure 11: Progressive construction for $\lceil (\Gamma^+ \cdot \mathsf{Notif\_T}^{(1)}) \rceil \mathbf{Rep}(C_3)$. Sub-figure (c) is obtained by applying $\mathsf{TSMC}_{\text{guard}}$ on sub-figure (b) and (a).

failures of $\mathbf{Rep}(C_3)$ propagate to the global verdict; once the guard closes, the open-ended obligation is discharged, and the monitor collapses to post acceptance provided that no violation has occurred. This example shows that the tight constructions scale to nested patterns of sequencing, repetition, guarding, and reparation while still preserving a direct correspondence with the prefix semantics of TACNL .

6.3 Forward-Looking Blaming Semantics

The tight semantics of TACNL identify when a contract is satisfied or violated, but do not explain *who* caused a violation. To attribute responsibility, we refine the violation verdicts based on which party failed to meet the relevant normative requirement. We break down tight and post violations into those caused by agent 1, those caused by agent 2, those caused jointly by both, and those where neither party is responsible (blameless cases). Formally, we introduce tight violation verdicts

$$\bot^t_1, \quad \bot^t_2, \quad \bot^t_{12}, \quad \bot^t_0,$$

and their post-violation counterparts

$$\bot^p_1, \quad \bot^p_2, \quad \bot^p_{12}, \quad \bot^p_0.$$

By replacing the undifferentiated violation verdicts of the five-valued semantics with these responsibility-aware variants, and keeping the three non-violating verdicts $?, \top^t, \top^p$, we obtain the forward-looking blame eleven-valued semantics

$$\mathbb{V}_{11} = \{?, \top^t, \top^p\} \cup \{\bot^t_S, \bot^p_S \mid S \in \{0, 1, 2, 12\}\}.$$

This refined judgement structure allows the monitors constructed in the next section to pinpoint the agents responsible for each contractual breach.

Blame Rules for Literals

Definition 37 (Blame assignment for literals)**.** *Let $p \in \{1,2\}$ be the main subject of the norm and let $\overline{p}$ denote the other party. Let $\langle A \rangle$ be a single-step word with $A \in \Gamma$. We write $a^{(i)} \in A$ when party i attempts a in this step.*

Obligation $\mathbf{O}_p(a)$. *Violation occurs if and only if the joint execution does not happen. Blame principle: if the subject does not attempt, blame the subject; otherwise, blame the other party for not cooperating:*

$$\langle A \rangle \vDash_{\bot^t_p} \mathbf{O}_p(a) \overset{\text{def}}{=} a^{(p)} \notin A,$$
$$\langle A \rangle \vDash_{\bot^t_{\overline{p}}} \mathbf{O}_p(a) \overset{\text{def}}{=} a^{(p)} \in A \wedge a^{(\overline{p})} \notin A.$$

These two cases partition a tight violation of $\mathbf{O}_p(a)$.

Prohibition $\mathbf{F}_p(a)$. *Violation requires the joint act to occur. Since the subject should refrain, blame the subject:*

$$\langle A \rangle \vDash_{\bot^t_p} \mathbf{F}_p(a) \overset{\text{def}}{=} \{a^{(1)}, a^{(2)}\} \subseteq A.$$

If only one agent attempts the action, it does not violate a prohibition, so no other possible blame arises. An agent cannot be blamed for a prohibition that was not assigned to it.

Power $\mathbf{P}_p(a)$. *Blame occurs only when the subject of the power attempts and the other party withholds cooperation, and the blame goes to the other party:*

$$\langle A\rangle \vDash_{\perp^t_{\overline{p}}} \mathbf{P}_p(a) \overset{\text{def}}{=} a^{(p)} \in A \wedge a^{(\overline{p})} \notin A.$$

Units: Valid ($\top$) and invalid ($\perp$). $\neg(\langle A\rangle \vDash_{\perp^t_S} \top)$ *for all S.* $\langle A\rangle \vDash_{\perp^t_0} \perp$ *by convention (unsatisfiable literal with no party subject).*

Post violation (prefix closure of blame). *Tight blame persists to extensions, and post blame is exactly "some earlier tight blame":*

$$\langle A\rangle \vDash_{\perp^t_S} \ell \implies \forall \pi \neq \varepsilon : \langle A\rangle\pi \vDash_{\perp^p_S} \ell, \qquad \pi \vDash_{\perp^p_S} \ell \iff \exists j < |\pi| : \pi[0,j] \vDash_{\perp^t_S} \ell.$$

Remark 3 (No joint blame at the literal level)**.** *Each literal is decided on a single step, and the only responsibility split is between the subject and the other party: either the subject fails to attempt, or the other party fails to cooperate, or no violation occurs. Hence, for literals, the blame set is always a singleton $S \in \{\{1\},\{2\}\}$ (or empty for $\perp$), never $\{12\}$.*

Example 26 (Obligation, prohibition, and power blame)**.** *By fixing $p = 1$, $\overline{p} = 2$. Consider the following letters $A \in \Gamma$:*

Obligation $\mathbf{O}_1(a)$.

$A = \emptyset$:	$\langle A\rangle \vDash_{\perp^t_1} \mathbf{O}_1(a)$	*(subject did not attempt)*
$A = \{a^{(2)}\}$:	$\langle A\rangle \vDash_{\perp^t_1} \mathbf{O}_1(a)$	*(subject did not attempt)*
$A = \{a^{(1)}\}$:	$\langle A\rangle \vDash_{\perp^t_2} \mathbf{O}_1(a)$	*(other party did not cooperate)*
$A = \{a^{(1)}, a^{(2)}\}$:	*no violation*	*(joint execution present).*

Prohibition $\mathbf{F}_1(a)$.

$A = \{a^{(1)}, a^{(2)}\}$:	$\langle A\rangle \vDash_{\perp^t_1} \mathbf{F}_1(a)$	*(subject should have refrained)*
$A = \{a^{(1)}\}$:	*no violation*	*The prohibited action was not successful*

Power $\mathbf{P}_1(a)$.

$A = \{a^{(1)}\}$:	$\langle A\rangle \vDash_{\perp^t_2} \mathbf{P}_1(a)$	*(subject asked, other party withheld)*
$A = \{a^{(1)}, a^{(2)}\}$:	*no violation*	*(properly supported)*
$A \in \{\emptyset, \{a^{(2)}\}\}$:	*no violation*	*(no unsupported subject attempt).*

Blame Propagation in Contracts

Conjunction. For $S \subseteq \{1,2\}$ of agent(s), and two contract C and C' from TACNL and a synchronous trace π, blame is defined for the conjunction $C \wedge C'$ is defined as:

$$\pi \vDash_{\perp^t_S} (C \wedge C') \iff \begin{cases} \pi \vDash_{\perp^t_S} C \text{ and } \pi \vDash_{\underline{\perp}} C', \\ \pi \vDash_{\perp^t_S} C' \text{ and } \pi \vDash_{\underline{\perp}} C, \\ \pi \vDash_{\perp^t_{S_1}} C \text{ and } \pi \vDash_{\perp^t_{S_2}} C' \text{ with } S = S_1 \cup S_2. \end{cases}$$

Where $\perp\!\!\!\perp$ stand for a non violation verdict, i.e, $\perp\!\!\!\perp \in \{?,\ \top^p,\ \top^t\}$
Intuition. The three cases summarize the possible outcomes of forward-looking blame. The blame goes to the agent responsible for the first violation of the contract: so either C or C', but both contracts could be violated at the same time point, in this case, the agent or agents responsible for *both simultaneous* violation get the blame.

For the rest of the operators, blame follows a similar definition as the tight violation, with $k \in [0, |\pi|]$:

Sequence. For $S \subseteq \{1,2\}$, contracts C, C' in TACNL , and a synchronous trace π:

$$\pi \vDash_{\perp^t_S} (C;C') \iff \begin{cases} \pi \vDash_{\perp^t_S} C, \\ \pi_k \vDash_{\top^t} C \text{ and } \pi^{k+1} \vDash_{\perp^t_S} C'. \end{cases}$$

Intuition. The first decisive failure before C has tightly succeeded belongs to C, so its blame propagates. Once C has tightly succeeded ($\top^t$) or is in post-success ($\top^p$), only C' can still fail, so the blame comes from C'. There is no tie, since C' becomes active only after C has tightly succeeded.

Reparation. For $S \subseteq \{1,2\}$, the blame for a reparation contract $C \blacktriangleright C'$ is defined as:

$$\pi \vDash_{\perp^t_S} (C \blacktriangleright C') \iff \exists k \text{ such that } \pi[0,k] \vDash_{\perp^t} C \text{ and } \pi[k{+}1, |\pi|] \vDash_{\perp^t_S} C'.$$

Intuition. A reparation clause becomes active only after a violation of C. The global blame set S, therefore, corresponds to the agents responsible for violating the reparation C' once it is triggered. The blame for C is not considered, as one cares only for the overall violation of the combined contracts.

Example 27 (Witness traces for all blame verdicts)**.** *We use* $\Sigma_C = \{\mathsf{PAY\_R}, \mathsf{PAY\_F}, \mathsf{OCC}\}$ *and letters* $A_t \subseteq \Gamma$ *with agent tags* $\cdot^{(1)}, \cdot^{(2)}$*. Recall*

$$C_2' := \mathbf{P}_1(\mathsf{OCC})\ ;\ \mathbf{P}_1(\mathsf{OCC}), \qquad C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F}).$$

Tight blame for agent 1

$$\pi_1 = \langle A_0 \rangle, \quad A_0 = \{\mathsf{OCC}^{(1)}\}$$

Here $\mathbf{P}_1(\mathsf{OCC})$ *and* $\mathbf{O}_1(\mathsf{PAY\_R})$ *are violated, the blame verdict are:*

- *The tenant (1) gets blamed for violating the obligation to pay rent:*
  $\pi_1 \vDash_{\perp^t_1} \mathbf{O}_1(\mathsf{PAY\_R})$.
- *The landlord (2) gets blamed for violating the power of the tenant to occupy the flat:*
  $\pi_1 \vDash_{\perp^t_2} \mathbf{P}_1(\mathsf{PAY\_R})$.

But the specification allows for the reparation $\mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$. *So consequently, no tight violation can be diagnosed at* $T = 1$*:*
$\pi_1 \models_? \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$. *Consequently, only the landlord gets the blame for the overall specification:*

$$\pi_1 \vDash_{\perp_2^t} C_2' \wedge C_3.$$

Moreover, consider the trace of $\pi_2 := \langle \{\mathsf{OCC}^{(1)}\}, \{\mathsf{OCC}^{(1)}\} \rangle$, *the extension of* π_1 *with the same event, as the blame is forward and tight looking, the blame is still assigned only to agent* 2 *(landlord) as they is responsible for the first violation.*

Let us consider instead the following trace $\pi_3 := \langle A_0', A_1 \rangle$ *with* $A_0' := \{\mathsf{OCC}^{(1)}, \mathsf{OCC}^{(2)}\}$ *and* $A_1 := \{\mathsf{OCC}^{(1)}\}$.

Here:

- *The landlord gets the blame at* $T = 2$ *for violating the power of the tenant to occupy the flat in the second month:*
  $\pi_3 \vDash_{\perp_2^t} \mathbf{P}_1(\mathsf{OCC}) \,;\, \mathbf{P}_1(\mathsf{OCC})$.
- *For the reparation clause* $\mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ *we must distinguish two different situations in which the fine is not honoured:*
  - *if the tenant never attempts to pay the fine, that is, no letter of the trace contains* $\mathsf{PAY\_F}^{(1)}$, *then the blame goes to agent 1:*
    $\pi \vDash_{\perp_1^t} \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$,
  - *if instead the tenant attempts to pay the fine and the landlord does not cooperate, for example, in a letter* A *with* $\mathsf{PAY\_F}^{(1)} \in A$ *and* $\mathsf{PAY\_F}^{(2)} \notin A$, *then the fine obligation is violated, and the blame goes to agent 2:*
    $\pi \vDash_{\perp_2^t} \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$.

6.4 From Tight Contract Satisfaction Monitor to Tight Blame Monitor

Definition 38 (Blame monitor)**.** *The* blame monitor, *written* $\mathcal{M}_{11}$, *is a Moore machine whose output alphabet is the eleven-valued blame verdict set* $\mathbb{V}_{11}$. *Formally,*

$$\mathcal{M}_{11} = (Q, q_0, \Gamma, \mathbb{V}_{11}, \delta, \lambda_{11}),$$

where:

1. *The output alphabet formed by 11 letters is*
$$\mathbb{V}_{11} = \{?, \top^t, \top^p\} \cup \{\perp_S^t, \perp_S^p \mid S \in \{0, 1, 2, 12\}\}.$$
2. Q *is the set of states and* $q_0 \in Q$ *is the initial state,*
3. $\Gamma = 2^\Sigma$ *is the input event alphabet,*
4. $\delta : Q \times \Gamma \to Q$ *is the transition function,*
5. $\lambda_{11} : Q \to \mathbb{V}_{11}$ *is the state output function.*

The blame monitor refines the five-valued tight satisfaction monitor by keeping the same control structure and replacing each violating region with a responsibility-aware verdict from $\mathbb{V}_{11}$.

Definition 39 (Blame monitor construction)**.** *Let C be a contract in* TACNL *. The* blame monitor construction *is a function on contracts, written* $\mathsf{BMC}(C)$*, that returns the blame monitor over $\mathbb{V}_{11}$ for C. We define* $\mathsf{BMC}(C)$ *by reusing the tight satisfaction monitor construction of Definition 28. Let*

$$\mathsf{TSMC}(C) = (Q, q_0, \Gamma, \mathbb{V}_5, \delta, \lambda_5)$$

be the tight satisfaction monitor for C. The corresponding blame monitor is

$$\mathsf{BMC}(C) := (Q, q_0, \Gamma, \mathbb{V}_{11}, \delta, \lambda_{11}),$$

that is, the state space, initial state, input alphabet, and transition function are reused from $\mathsf{TSMC}(C)$*, and only the output function is refined from λ_5 to λ_{11} as described below.*

Lifting contract verdicts to blame verdicts. Intuitively, λ_{11} refines the five-valued verdicts of the tight contract monitor by attaching a blame set to every violating region. The three non-violating outcomes,

$$?, \quad \top^t, \quad \top^p,$$

are kept unchanged. Whenever the tight semantics reach a tight violation at some prefix, the corresponding state in the blame monitor outputs a symbol of the form $\bot^t_S$ where $S \in \{\{1\},\{2\},\{1,2\},\emptyset\}$ specifies who is responsible? Likewise, every post-violation region is labeled by some $\bot^p_S$.

Formally, let $\vDash_{\bot^t_S}$ and $\vDash_{\bot^p_S}$ be the denotational blame judgments introduced above. For a finite trace π and prefix index $k < |\pi|$, define the *ideal* blame verdict

$$\mathsf{Blame}(C, \pi[0,k]) \in \mathbb{V}_{11}$$

as follows:

$$\mathsf{Blame}(C, \pi[0,k]) = \begin{cases} ? & \text{if } \pi[0,k] \models_{?} C, \\ \top^t & \text{if } \pi[0,k] \models_{\top^t} C, \\ \top^p & \text{if } \pi[0,k] \models_{\top^p} C, \\ \bot^t_S & \text{if } \pi[0,k] \vDash_{\bot^t_S} C, \\ \bot^p_S & \text{if } \pi[0,k] \vDash_{\bot^p_S} C. \end{cases}$$

By construction of the five-way semantics and the blame rules, exactly one of these cases applies to each prefix, and the set S is uniquely determined whenever a blame judgement holds.

Definition 40 (Blame refinement of a contract monitor)**.** *Let $\mathcal{M}(C) = (Q, \Gamma, \delta, q_0, \lambda)$ be the five-valued Moore monitor for contract C, with outputs in $\mathbb{V}_5 = \{?, \top^t, \bot^t, \top^p, \bot^p\}$.*

The tight blame monitor $\mathcal{BM}(C)$ *is constructed by retaining the control structure of $\mathcal{M}(C)$ while refining the violation outputs to pinpoint responsibility. Formally:*

$$\mathcal{BM}(C) = (Q, \Gamma, \delta, q_0, \lambda^{\mathcal{BM}}),$$

where the new output function $\lambda^{\mathcal{BM}} : Q \to \mathbb{V}_{11}$ is defined for every state $q \in Q$ (reached by some trace π) as follows:

$$\lambda^{\mathcal{BM}}(q) = \begin{cases} \lambda(q) & \text{if } \lambda(q) \in \{?,\ \top^t,\ \top^p\}, \\ \bot^t_S & \text{if } \lambda(q) = \bot^t \text{ and } \pi \vDash_{\bot^t_S} C, \\ \bot^p_S & \text{if } \lambda(q) = \bot^p \text{ and } \pi \vDash_{\bot^p_S} C. \end{cases}$$

Theorem 6.2 (Correctness and Consistency of the Blame Monitor)**.** *Let C be a contract in* TACNL . *Let $\mathcal{M}^{TS}(C)$ be its tight satisfaction monitor with output function λ_5, and let $\mathcal{BM}(C)$ be its blame monitor with output function $\lambda^{\mathcal{BM}}$ (also denoted λ_{11}). Let* $\mathsf{Blame}(C,\pi)$ *denote the denotational blame verdict of C on trace π as defined in the forward-looking semantics (Subsection.6.1).*

For every finite trace π, the following equality holds:

$$\lambda^{\mathcal{BM}}\big(\delta^{\mathcal{BM}}(q_0,\pi)\big) \;=\; \mathsf{Blame}(C,\pi).$$

Furthermore, the blame monitor is consistent with the tight satisfaction monitor for non-violating verdicts. For all traces π:

$$\mathsf{Blame}(C,\pi) \in \{?, \top^t, \top^p\} \implies \lambda^{\mathcal{BM}}\big(\delta^{\mathcal{BM}}(q_0,\pi)\big) \;=\; \lambda_5\big(\delta^{\mathcal{M}^{TS}}(q_0,\pi)\big).$$

Proof. The proof proceeds by structural induction on C. We define $\lambda^{\mathcal{BM}}$ (denoted λ_{11}) using λ_5 and verify both correctness and consistency for each operator.

Base Case: Literals (ℓ). We defined $\lambda_{11}^{lit}(q,A)$ such that if $\lambda_5(q) \in \{?, \top^t, \top^p\}$, then $\lambda_{11}^{lit}(q,A) = \lambda_5(q)$. If $\lambda_5(q) = \bot^t$, it maps to $\bot^t_S$ (where $S \neq \emptyset$). Thus, $\lambda_{11}(q) \in \{?, \top^t, \top^p\} \iff \lambda_5(q) \in \{?, \top^t, \top^p\}$ and the values are identical. Consistency holds. Correctness holds by Definition 40.

Inductive Step: Conjunction ($C_1 \wedge C_2$). Let $q = (q_1, q_2)$. The definition of $\lambda_{11}^{\wedge}$ defaults to the combination table $\lambda_5^{\mathsf{comb}}$ whenever neither component outputs a blame verdict (which corresponds to neither component outputting $\bot^t$ or $\bot^p$).

$$\lambda_{11}^{\wedge}(q) = \lambda_5^{\mathsf{comb}}(\lambda_5(q_1), \lambda_5(q_2)) \quad \text{if no blame detected.}$$

Since $\lambda_5^{\wedge}$ is defined exactly by this table, and blame verdicts $\bot^t_S$ are only introduced when at least one sub-monitor has a violation, the non-violating outcomes are identical. $\lambda^{\mathcal{BM}}(q) = \top^t \iff \lambda_5(q) = \top^t$ (and similarly for $\top^p, ?$).

Inductive Step: Sequence ($C_1;C_2$). The state space is partitioned into Q_1 and Q_2.

- If $q \in Q_1$: $\lambda^{;}_{11}(q) = \lambda^{1}_{11}(q)$. By IH, λ^{1}_{11} is consistent with λ^{1}_{5}. Since $\lambda^{;}_{5}(q) = \lambda^{1}_{5}(q)$ here, consistency is preserved.
- If $q \in Q_2$: $\lambda^{;}_{11}(q) = \lambda^{2}_{11}(q)$. By IH, λ^{2}_{11} is consistent with λ^{2}_{5}. Since $\lambda^{;}_{5}(q) = \lambda^{2}_{5}(q)$ here, consistency is preserved.

Inductive Step: Reparation ($C_1 \blacktriangleright C_2$). The state space is $Q_1 \cup Q_2$.

- If $q \in Q_1$: The construction ensures q is non-violating for C_1. We defined $\lambda^{\blacktriangleright}_{11}(q) = \lambda^{1}_{5}(q)$. Since $\lambda^{\blacktriangleright}_{5}(q) = \lambda^{1}_{5}(q)$, they are identical.
- If $q \in Q_2$: The primary contract failed. We defined $\lambda^{\blacktriangleright}_{11}(q) = \lambda^{2}_{11}(q)$. By IH, this is consistent with $\lambda^{2}_{5}(q)$. Since $\lambda^{\blacktriangleright}_{5}(q) = \lambda^{2}_{5}(q)$, consistency is preserved.

Conclusion. For all constructions, $\lambda^{\mathcal{BM}}(q) = \lambda_5(q)$ whenever $\lambda_5(q)$ is a non-violating verdict. Whenever $\lambda_5(q)$ is a violation $(\bot^t, \bot^p)$, $\lambda^{\mathcal{BM}}(q)$ refines it to a blame verdict $(\bot^t_S, \bot^p_S)$. Thus, the monitor is consistent for satisfaction/undecided verdicts and correct for blame assignment. ☐

This transformation effectively partitions the set of generic violation states into disjoint subsets of blamed states:

- *The tight violation states are split:* $\{q \mid \lambda(q) = \bot^t\} = \bigcup_S \{q \mid \lambda^{\mathcal{BM}}(q) = \bot^t_S\}$,
- *The post violation states are split:* $\{q \mid \lambda(q) = \bot^p\} = \bigcup_S \{q \mid \lambda^{\mathcal{BM}}(q) = \bot^p_S\}$.

Proof of Theorem 6.2: Definition of λ_{11}. The proof proceeds by structural induction on the contract C. We construct the blame output function λ_{11} for the blame monitor $\mathcal{BM}(C)$ by refining the output function λ_5 of the tight satisfaction monitor $\mathsf{TSMC}(C)$.

Base Case: Literals. Let $\ell = \mathbf{O}_p(a)$. The 5-valued monitor has a tight violation state q_v where $\lambda_5(q_v) = \bot^t$. The blame monitor refines this output based on the input letter A that triggered the transition to q_v. We define $\lambda^{lit}_{11}(q,A)$ as:

$$\lambda^{lit}_{11}(q,A) = \begin{cases} \bot^t_p & \text{if } \lambda_5(q) = \bot^t \text{ and } a^{(p)} \notin A, \\ & \text{(Subject failed to attempt)} \\ \bot^t_{\bar{p}} & \text{if } \lambda_5(q) = \bot^t \text{ and } a^{(p)} \in A \wedge a^{(\bar{p})} \notin A, \\ & \text{(Counterparty withheld cooperation)} \\ \bot^p_S & \text{if } \lambda_5(q) = \bot^p \text{ (inherits previous blame } S\text{)}, \\ \lambda_5(q) & \text{if } \lambda_5(q) \in \{?, \top^t, \top^p\}. \end{cases}$$

This matches the blame assignment for literals defined in Definition 40.

Inductive Step: Conjunction. Let $C = C_1 \wedge C_2$. The monitor state is (q_1, q_2). We define $\lambda_{11}^{\wedge}$ using the blame functions $\lambda_{11}^1, \lambda_{11}^2$ of the sub-monitors and their 5-valued checks.

$$\lambda_{11}^{\wedge}(q_1,q_2) = \begin{cases} \bot^t_{S_1 \cup S_2} & \text{if } \lambda_{11}^1(q_1) = \bot^t_{S_1} \text{ and } \lambda_{11}^2(q_2) = \bot^t_{S_2}, \\ \bot^t_{S_1} & \text{if } \lambda_{11}^1(q_1) = \bot^t_{S_1} \text{ and } \lambda_5^2(q_2) \in \{?, \top^t, \top^p\}, \\ \bot^t_{S_2} & \text{if } \lambda_{11}^2(q_2) = \bot^t_{S_2} \text{ and } \lambda_5^1(q_1) \in \{?, \top^t, \top^p\}, \\ \bot^p_{S_1 \cup S_2} & \text{if } \lambda_{11}^1(q_1) = \bot^p_{S_1} \text{ or } \lambda_{11}^2(q_2) = \bot^p_{S_2}, \\ \lambda_5^{\mathsf{comb}}(\lambda_5^1(q_1), \lambda_5^2(q_2)) & \text{otherwise.} \end{cases}$$

This implements the conjunction blame propagation rules.

Inductive Step: Sequence. Let $C = C_1; C_2$. The state space is partitioned into Q_1 (active C_1) and Q_2 (active C_2).

$$\lambda_{11}^{;}(q) = \begin{cases} \lambda_{11}^1(q) & \text{if } q \in Q_1, \\ \lambda_{11}^2(q) & \text{if } q \in Q_2. \end{cases}$$

Since Q_1 contains only states where C_1 has not yet succeeded, any violation here is attributed to C_1. Once in Q_2, C_1 has succeeded, so blame falls on C_2.

Inductive Step: Reparation. Let $C = C_1 \blacktriangleright C_2$. The state space is partitioned into Q_1 (active C_1) and Q_2 (active reparation C_2).

$$\lambda_{11}^{\blacktriangleright}(q) = \begin{cases} \lambda_5^1(q) & \text{if } q \in Q_1, \\ \lambda_{11}^2(q) & \text{if } q \in Q_2. \end{cases}$$

By construction, Q_1 excludes all violation states of C_1, so λ_{11} simply returns the non-violating 5-valued verdict. If C_1 fails, the monitor moves to Q_2, where the blame is determined entirely by the reparation contract C_2. □

In all cases, λ_{11} correctly maps the state to the specific blame verdict defined by the forward-looking semantics. We now move to illustrate this refinement with two interesting examples.

Example 28 (Blame Monitor for $C_2 \wedge C_3$)**.** *Let us recall that* $C_2 = \mathbf{P}_1(\mathsf{OCC})$ *represents the tenant's power to occupy the property, and* $C_3 = \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ *represents the obligation to pay rent, repaired by paying a fine. The following figure shows the blame refinement of the monitor in Fig. 10d. The generic violation state is partitioned into specific blame verdicts based on the cause of the failure.*

Although the previous example is constructed using a conjunction, the reparation operator within C_3 delays the assignment of blame for the payment obligation. Specifically, if the tenant fails to pay rent, the monitor transitions to a waiting state for the repair (outputting ?) rather than immediately emitting a violation verdict. Consequently,

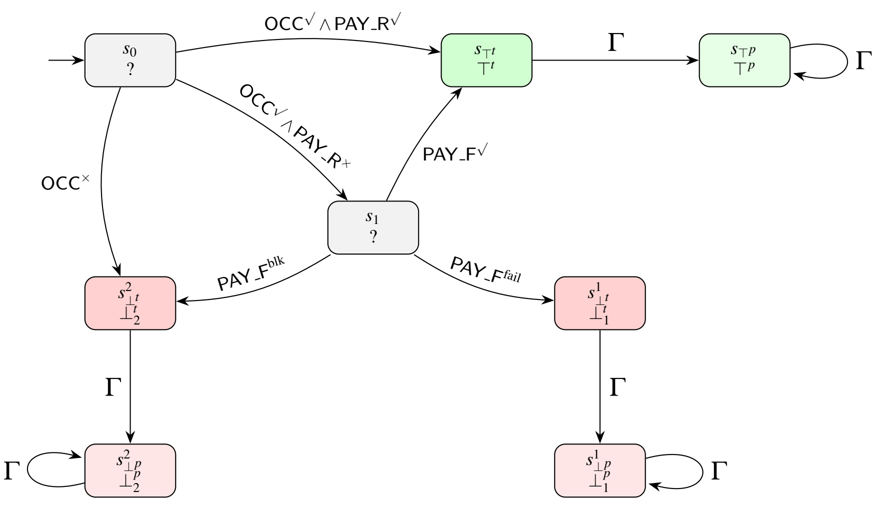

s_0 ?
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\checkmark}$
$s_{\top^t}$ $\top^t$
Γ
$s_{\top^p}$ $\top^p$
Γ
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}_{+}$
$\mathsf{PAY\_F}^{\checkmark}$
$\mathsf{OCC}^{\times}$
s_1 ?
$\mathsf{PAY\_F}^{\mathrm{blk}}$
$\mathsf{PAY\_F}^{\mathrm{fail}}$
$s^2_{\perp^t}$ $\perp^t_2$
$s^1_{\perp^t}$ $\perp^t_1$
Γ
Γ
Γ
$s^2_{\perp^p}$ $\perp^p_2$
$s^1_{\perp^p}$ $\perp^p_1$
Γ

Figure 12: Blame Monitor $\mathcal{BM}(C_2 \wedge C_3)$. **Changes from Tight Monitor:** The state $s_{\perp^t}$ is split into $s^2_{\perp^t}$ (Landlord blame) and $s^1_{\perp^t}$ (Tenant blame). **Edge Definitions:** $\mathsf{OCC}^{\times}$: Tenant attempts OCC, Landlord blocks. $\mathsf{PAY\_F}^{\mathrm{fail}}$: Tenant does not attempt $\mathsf{PAY\_F}$ ($\mathsf{PAY\_F}^{(1)} \notin A$). $\mathsf{PAY\_F}^{\mathrm{blk}}$: Tenant attempts $\mathsf{PAY\_F}$, Landlord blocks.

it is impossible for both conjuncts to return a tight violation $\perp^t$ at the same initial step. To illustrate a scenario where the monitor can output the joint blame verdict $\perp^t_{12}$, we consider the reparation-free reduction of the specification: $C_2 \wedge \mathbf{O}_1(\mathsf{PAY\_R})$.

Example 29 (Blame Monitor with double blame)**.** *The following monitor shows the emergence of joint blame. From the initial state s_0, three distinct violation paths are possible depending on who fails. The path to $s^{12}_{\perp^t}$ represents the simultaneous failure of both parties.*

Example 30 (Blame Monitor for $\mathbf{Rep}(C_3)$)**.** *The figure below shows the blame monitor for the unbounded repetition of the rent-and-reparation contract. The generic violation state $q_{\perp^t}$ from the standard monitor is split into $s^1_{\perp^t}$ and $s^2_{\perp^t}$. Crucially, once the monitor transitions to a post-violation sink (e.g., $s^1_{\perp^p}$), it loops on any input $*$. This demonstrates the "first blame" limitation: if the tenant is blamed for missing a fine, the monitor will never blame the landlord for any future misconduct.*

6.5 Conclusion and Limitation

We have presented forward-looking semantics and a corresponding monitor construction that refine the standard satisfaction verdicts by assigning responsibility. This approach is computationally efficient and provides immediate feedback on the *status* of the contract, allowing for runtime enforcement and dispute resolution at the moment a breach occurs.

However, this prefix-based view naturally implies a limitation regarding the completeness of the violation history. The semantics is designed to detect the *first* decisive

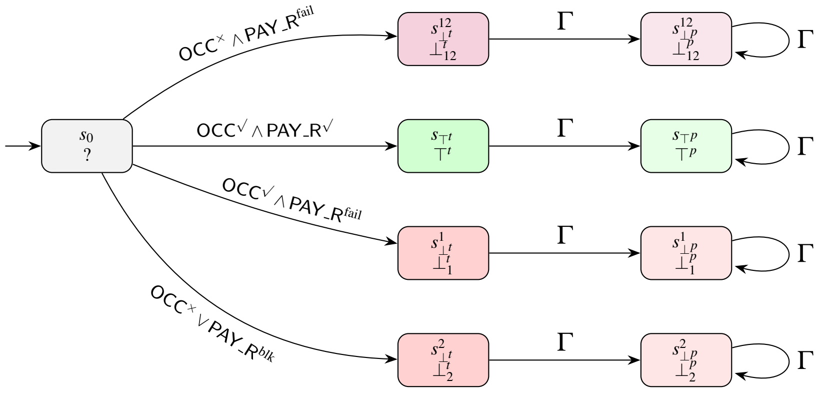

s_0
?
$\mathsf{OCC}^{\times} \wedge \mathsf{PAY\_R}^{\text{fail}}$
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\checkmark}$
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\text{fail}}$
$\mathsf{OCC}^{\times} \vee \mathsf{PAY\_R}^{blk}$
$s^{12}_{\perp^t}$ $\perp^t_{12}$
$s^{12}_{\perp^p}$ $\perp^p_{12}$
$s_{\top^t}$ $\top^t$
$s_{\top^p}$ $\top^p$
$s^1_{\perp^t}$ $\perp^t_1$
$s^1_{\perp^p}$ $\perp^p_1$
$s^2_{\perp^t}$ $\perp^t_2$
$s^2_{\perp^p}$ $\perp^p_2$
Γ

Figure 13: Blame Monitor for $C_2 \wedge \mathbf{O}_1(\mathsf{PAY\_R})$.
Edge Definitions:
$\mathsf{PAY\_R}^{\text{fail}}$: Tenant does not attempt payment ($\mathsf{PAY\_R}^{(1)} \notin A$).
$\mathsf{OCC}^{\times}$: Tenant attempts occupation, Landlord blocks.
$\mathsf{PAY\_R}^{\text{blk}}$: Tenant attempts to pay and Landlord blocks.
The state $s^{12}_{\perp^t}$ is reached only when both violations occur in the same step.

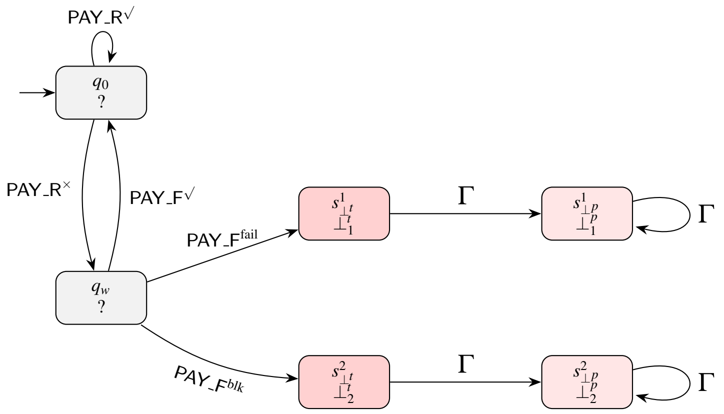

$\mathsf{PAY\_R}^{\checkmark}$
q_0
?
$\mathsf{PAY\_R}^{\times}$
$\mathsf{PAY\_F}^{\checkmark}$
$\mathsf{PAY\_F}^{\text{fail}}$
q_w
?
$\mathsf{PAY\_F}^{blk}$
$s^1_{\perp^t}$ $\perp^t_1$
$s^1_{\perp^p}$ $\perp^p_1$
$s^2_{\perp^t}$ $\perp^t_2$
$s^2_{\perp^p}$ $\perp^p_2$
Γ

Figure 14: Blame Monitor for $\mathbf{Rep}(C_3)$. **Limitation:** If the trace reaches $s^1_{\perp^p}$ (Tenant blame), the monitor remains there forever. Even if the landlord subsequently blocks a valid payment attempt ($\mathsf{PAY\_F}^{\text{blk}}$) in a future step, the verdict remains $\perp^p_1$.

violation that renders the contract permanently unsatisfiable. Once the monitor transitions to a post-violation sink state ($\perp^p_S$), the verdict becomes immutable. A practical consequence of this property is that, even when processing infinite words or streams, the monitor can be programmed to halt execution immediately after the first tight violation is detected, as no future event can alter the blame assignment. Consequently, any subsequent violations committed by other agents at later time steps are effectively masked by the first failure.

This limitation is particularly evident in open-ended contracts involving the repetition operator. As illustrated by the monitor for $\mathbf{Rep}(C_3)$ (Figure 14), if the tenant is blamed for failing to pay the reparation in one cycle, the monitor enters the sink state $s^1_{\perp p}$. Even if the interaction continues and the landlord subsequently violates their permission or obligation in a future cycle (e.g., by blocking a valid payment attempt), the monitor remains fixed on the initial verdict $\perp^p_1$. Therefore, while this framework is sufficient for determining *why* a contract failed, it does not support a cumulative accounting of *all* violations that occur throughout the lifespan of a long-running interaction. This is a notable constraint, as in law and normative systems, one typically has to account for all violations to determine the full extent of liability or penalties. Capturing such multi-point violations would require a mechanism to reset or parallelize monitoring threads after a failure, rather than absorbing them into a permanent sink.

Finally, extending this framework to support a cumulative blame semantics suggests interesting theoretical challenges. In particular, the interaction between timed operators and conjunctions complicates fault aggregation. For instance, determining whether overlapping failures in concurrent branches or repeated violations within sliding time windows should be counted as distinct or continuous breaches requires a more complex, possibly non-monotonic, judgment structure than the one presented here.

7 Quantitative Violation Semantics

7.1 Motivation and Scope

Beyond Boolean Verdicts. The basic restriction of the forward-looking tight satisfaction semantics, as defined in the previous section, resides in its binary and prefix-closed nature. In that framework, the monitoring process halts definitively at the first tight violation. Despite being computationally efficient for stopping compliance systems (e.g., an access control system that is failing), this "first-fail" approach is insufficient for post-hoc auditing or dispute resolution. It masks the full history of non-compliance, failing to capture cumulative violations or distinct failures by multiple agents over time—a critical requirement for comprehensive legal responsibility. To tackle this limitation, it is necessary to transition from a boolean verdict to a *quantitative semantics*, changing the emphasis from the question "Did a violation occur?" to "How many violations occurred, and what was their magnitude?"

The Challenge of Temporal Scope. A prerequisite for counting violations is defining the temporal window over which the contract is evaluated. In contrast to traditional model checking, which commonly analyzes systems that are supposed to run forever, contract monitoring operates on finite, evolving prefixes. We identify two primary strategies for defining this scope:

1. **Static Pre-computation:** One might attempt to calculate a fixed duration T for a contract C. However, contracts containing unbounded repetitions ($\mathbf{Rep}(C)$) or input-dependent regular expressions (guards/triggers) do not have a statically determinable length.

2. **Dynamic On-the-Fly Detection:** The alternative is to determine the contract's status dynamically at every step. Here, the monitor continuously computes a *residual contract*—a formula representing what remains to be fulfilled given the history of events.

This work adopts the **Dynamic On-the-Fly** approach. This requires a structural tracking mechanism, which we formalize as the *Contract Progress Monitor* (CPM). The CPM acts as a derivative function: given a contract C and an incoming event, it computes the contract C' that must be enforced in the next time step. Isolating the state of the contract (via the CPM) from the evaluation of compliance (via a scoring function) enables a modular framework that can track active duties, handle contrary-to-duty (CTD) transitions, and attribute blame with precision over time. The scoring function is critical in this framework, as it provides a quantitative evaluation of compliance. This approach not only determines whether obligations have been met, but also assesses the degree of compliance or non-compliance. The interaction with the CPM measures the magnitude of violations, presenting an in-depth and nuanced view of contractual adherence that improves accountability and enforcement precision.

7.2 Contract Progress Monitor

The core of our measurement framework is the *progression function*, Prog. Conceptually similar to the Brzozowski derivative [5] for regular expressions, Prog consumes the current event and the current contract to produce the *residual contract* for the subsequent step.

Definition 41 (Contract Progression Function)**.** *Let $\Gamma = 2^{\Sigma}$ be the event alphabet. We extend the set of contracts $\mathcal{C}$ with a distinguished symbol ε_C, representing a* discharged contract *(one that implies no further obligations).*

The progression function $\mathsf{Prog} : \Gamma^* \times \mathsf{TACNL} \to \mathsf{TACNL} \cup \{\varepsilon_C\}$ *is defined recursively. For the empty trace ε,* $\mathsf{Prog}(\varepsilon, C) := C$*. For a single event step $\langle A \rangle$ with $A \in \Gamma$, the function is defined on the structure of C as follows.*

Literals (State Update). *For any literal ℓ (obligation, permission, or prohibition):*

$$\mathsf{Prog}(\langle A \rangle, \ell) := \varepsilon_C$$

Rationale: *Structurally, a literal applies to a single time step. Once the A step occurs, the literal is consumed. Whether A satisfied or violated ℓ is immaterial to the* progression *(the duty is passed); the violation is recorded separately by the quantitative scoring function defined later.*

Conjunction (Parallel Progress).

$$\mathsf{Prog}(\langle A \rangle, C_1 \wedge C_2) := \mathsf{Prog}(\langle A \rangle, C_1) \wedge \mathsf{Prog}(\langle A \rangle, C_2)$$

We assume the symbolic identity where ε_C is the neutral element for conjunction: $\varepsilon_C \wedge C' \equiv C'$.

Sequence (Sequential Handover). *For a sequence $C_1;C_2$, progression determines if the current step concludes C_1:*

$$\mathsf{Prog}(\langle A\rangle, C_1;C_2) := \begin{cases} \mathsf{Prog}(\langle A\rangle, C_1);C_2 & \textit{if } \mathsf{Prog}(\langle A\rangle, C_1) \neq \varepsilon_C, \\ \mathsf{Prog}(\langle A\rangle, C_2) & \textit{if } \mathsf{Prog}(\langle A\rangle, C_1) = \varepsilon_C. \end{cases}$$

If C_1 is discharged by step A (i.e., its residual is ε_C), the monitor immediately activates the first step of the next portion C_2.

Reparation (Contrary-to-Duty Branching). *The reparation construct is unique because its structural progression depends on the satisfaction of the primary obligation. This is the only case where* Prog *relies on the tight satisfaction relation ($\models_{\top^t}, \models_{\bot^t}, \models_?$) to determine the path:*

$$\mathsf{Prog}(\langle A\rangle, C_1 \blacktriangleright C_2) := \begin{cases} \mathsf{Prog}(\langle A\rangle, C_1) \blacktriangleright C_2 & \textit{if } \langle A\rangle \models_? C_1 \textit{ (Pending)}, \\ \mathsf{Prog}(\langle A\rangle, C_2) & \textit{if } \langle A\rangle \models_{\bot^t} C_1 \textit{ (Violation} \rightarrow \textit{Repair)}, \\ \varepsilon_C & \textit{if } \langle A\rangle \models_{\top^t} C_1 \textit{ (Satisfaction} \rightarrow \textit{Discharge)}. \end{cases}$$

If a violation occurs ($\langle A\rangle \models_{\bot^t} C_1$), the primary contract is discarded, and the secondary contract C_2 is activated immediately for the next *step.*

Repetition. *Repetition unrolls the contract one step at a time:*

$$\mathsf{Prog}(\langle A\rangle, \mathbf{Rep}(C)) := \mathsf{Prog}(\langle A\rangle, C); \mathbf{Rep}(C)$$

Guarded and Triggered Contracts. *These constructs rely on regular expression matching.*

$$\mathsf{Prog}(\langle A\rangle, \lceil re\rceil C) := \begin{cases} \varepsilon_C & \textit{if } \langle A\rangle \models_{\top^t} \lceil re\rceil C \textit{ (Guard satisfies, release)}, \\ \lceil \mathsf{Prog}_{re}(\langle A\rangle, re)\rceil \mathsf{Prog}(\langle A\rangle, C) & \textit{otherwise (Guard persists)}. \end{cases}$$

$$\mathsf{Prog}(\langle A\rangle, \langle\langle re\rangle\rangle C) := \begin{cases} \varepsilon_C & \textit{if } \langle A\rangle \models_{\bot^t} re \textit{ (Trigger failed permanently)}, \\ C & \textit{if } \langle A\rangle \models_{\top^t} re \textit{ (Trigger fires)}, \\ \langle\langle \mathsf{Prog}_{re}(\langle A\rangle, re)\rangle\rangle C & \textit{if } \langle A\rangle \models_? re \textit{ (Trigger pending)}. \end{cases}$$

Regular Expression Derivatives (Prog_{re}). *The helper function Prog_{re} computes the standard derivative [5] of the regular expression. For atomic sets A' and step A, $\mathsf{Prog}_{re}(\langle A\rangle, A') = \varepsilon$ if matches, else failure. For operations like union ($|$) and Kleene plus ($^+$), the derivatives follow standard automata-theoretic constructions.*

Lemma 17 (Termination)**.** *For each finite prefix π, $\mathsf{Prog}(\pi, C)$ terminates in at most $|\pi|$ recursive steps.*

To illustrate the CPM, the evolution of the residual contract under different traces is examined. The evaluation initiates with the fundamental building block of normative enforcement: the reparation.

Example 31 (Progression of Reparation)**.** *Consider the basic rental reparation clause:*

$$C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F}).$$

Scenario 1: Compliance. *In the first trace π, the tenant pays rent in Month 1 ($\mathsf{PAY\_R} \in A_1$). Since $\langle A_1 \rangle \models_{\top'} \mathbf{O}_1(\mathsf{PAY\_R})$, the condition for discharge is met. The reparation structure collapses to ε_C, meaning no further obligations exist for Month 2.*

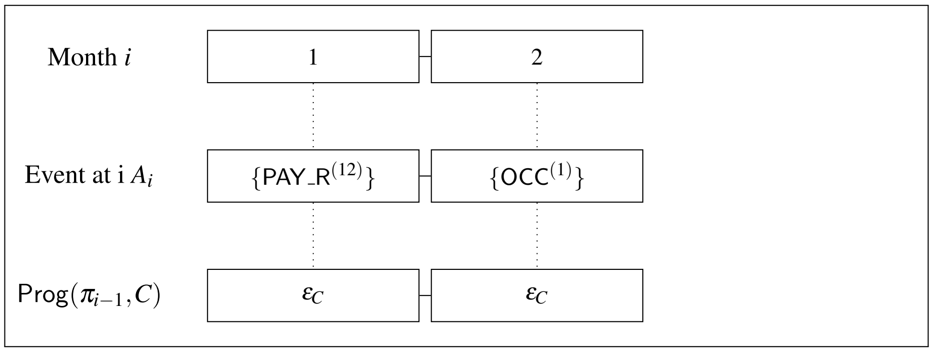


Figure 15: Progression on $\mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ with a trace for which no reparation is not required

Scenario 2: Violation and Repair. *In trace π', the tenant fails to pay in Month 1 ($\mathsf{PAY\_R} \notin A'_1$). Here, $\langle A'_1 \rangle \models_{\perp'} \mathbf{O}_1(\mathsf{PAY\_R})$. Consequently, the CPM activates the repair branch. The residual for Month 2 becomes $\mathbf{O}_1(\mathsf{PAY\_F})$, obliging the tenant to pay the fine.*

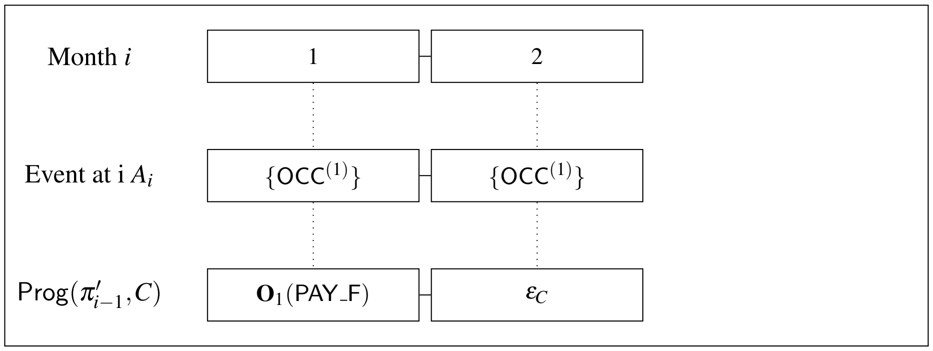


Figure 16: Progression on $\mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ with a trace for which no reparation is not required

After establishing the evolution of single-step reparations, the analysis is extended to ongoing contracts. The following example demonstrates how the CPM handles infinite streams with recurring duties, showing how violations in one period persist into subsequent periods.

Example 32 (Progression of Infinite Repetition)**.** *Consider the recurring contract* $\mathbf{Rep}(C_3)$*, where the tenant must pay rent (or a fine) every month.*

$$\mathbf{Rep}(C_3) = \mathbf{Rep}(\mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})).$$

Trace Analysis. *Consider a trace in which the tenant pays in Month 1 but fails to pay in Month 2, instead occupying the property.*

- ***Step 1** (A_1)**:*** *The tenant pays. The instance of C_3 for Month 1 is discharged. Due to the repetition operator, the residual is* $\varepsilon_C; \mathbf{Rep}(C_3) \equiv \mathbf{Rep}(C_3)$*. The contract effectively "resets" for the next month.*
- ***Step 2** (A_2)**:*** *The tenant occupies but does not pay. The instance of C_3 for Month 2 violates the condition. Unlike Step 1, the residual does not reset cleanly. Instead, the violated obligation transforms into its reparation* $\mathbf{O}_1(\mathsf{PAY\_F})$*, which must be fulfilled in the* next *step (Month 3), alongside the continuing repetition* $\mathbf{Rep}(C_3)$*.*

This results in an accumulation of duties: the fine from Month 2 and the new rent for Month 3.

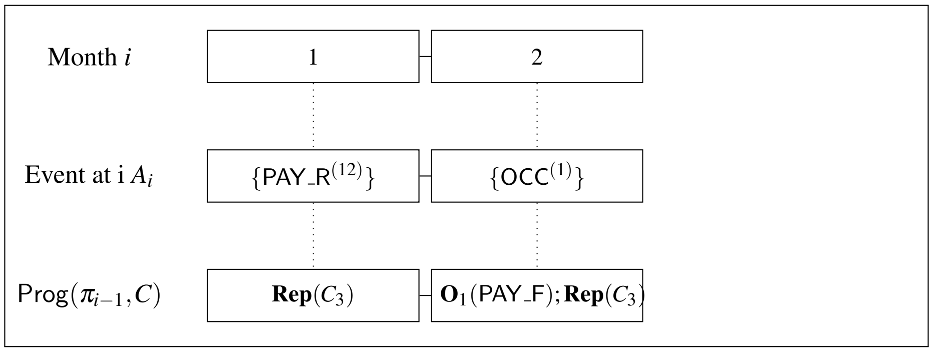

Month i
1
2
Event at i A_i
$\{\mathsf{PAY\_R}^{(12)}\}$
$\{\mathsf{OCC}^{(1)}\}$
$\mathsf{Prog}(\pi_{i-1}, C)$
$\mathbf{Rep}(C_3)$
$\mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3)$

Figure 17: Progression on $\mathbf{Rep}(C_3)$ where the obligation is met in the first month but violated in the second.

While repetitions capture simple recurring duties, more complex contracts are often bound by conditions. The following analysis studies how the CPM handles *guarded contracts*, in which the outer structure (the guard) and the inner structure (the obligations) evolve independently until a termination event occurs.

Example 33 (Progression of Guarded Contracts)**.** *A guarded contract that persists until a termination notice (*$\mathsf{Notif\_T}$*) is issued is examined.*

$$\lceil \Gamma^{+} \cdot \mathsf{Notif\_T}^{(1)} \rceil \mathbf{Rep}(C_3)$$

Trace 1: Successful Termination. *The tenant pays in Month 1 (A_1) and issues a termination notice in Month 2 (A_2).*

- *At $i = 1$, the event A_1 satisfies the inner contract C_3 (rent paid), but does not satisfy the guard (no notice). The residual is the guarded repetition.*
- *At $i = 2$, the event A_2 contains* $\mathsf{Notif\_T}$. *This satisfies the guard expression. The CPM immediately reduces the entire contract to ε_C, signifying the contract has ended.*

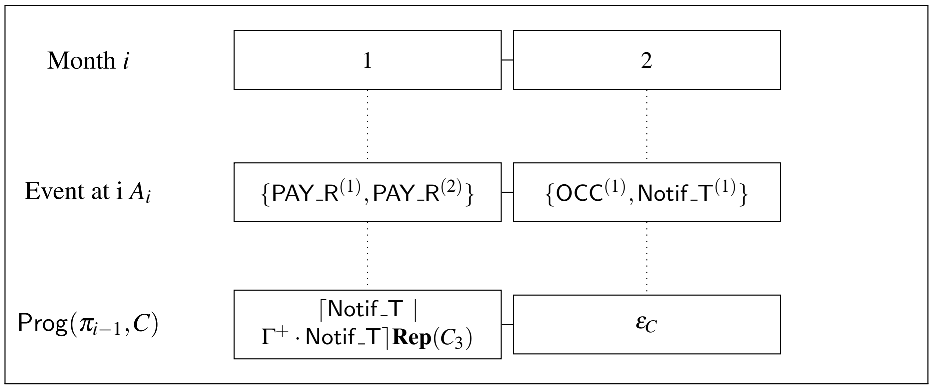

Month i
1
2
Event at i A_i
$\{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}$
$\{\mathsf{OCC}^{(1)}, \mathsf{Notif\_T}^{(1)}\}$
$\mathsf{Prog}(\pi_{i-1}, C)$
$\lceil \mathsf{Notif\_T} \mid \Gamma^{+} \cdot \mathsf{Notif\_T} \rceil \mathbf{Rep}(C_3)$
ε_C

Figure 18: Progression on guarded contract where the termination notice at step 2 discharges the contract.

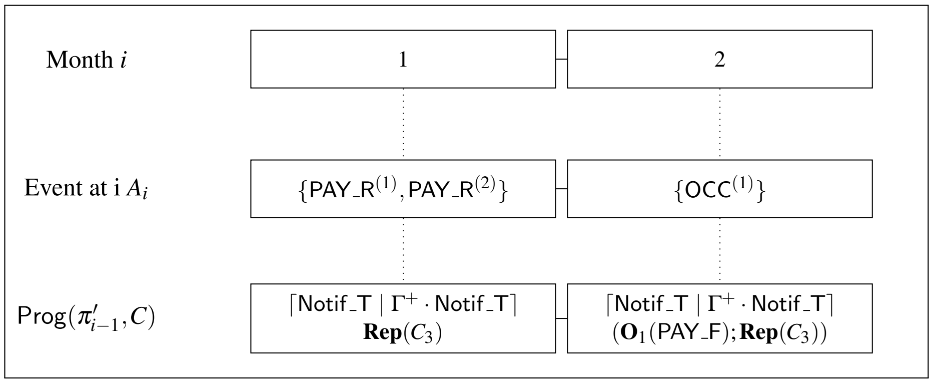
Trace 2: Pending Guard with Internal Violation. *In this scenario, the tenant pays in Month 1 but fails to pay in Month 2 (and gives no notice).*

- *At $i = 2$, the guard is* not *satisfied.*
- *Simultaneously, the inner contract $\mathbf{Rep}(C_3)$ processes the event. Since rent was not paid, the inner contract evolves into a reparation state ($\mathbf{O}_1(\mathsf{PAY\_F})$).*
- *The resulting residual is a guarded reparation:* $\lceil \dots \rceil (\mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3))$.

This illustrates how the CPM maintains the "wrapper" (the guard) while the content inside (the obligations) evolves and accumulates violations independently.

7.3 Quantitative Monitoring of Contract Compliance

To define the quantitative violation semantics that do not stop at the first violation prefix, we reuse the *Contract Progress function* as the underlying state-transition mechanism. The progression function Prog dynamically evolves the contract after each observation, producing a sequence of *residual contracts* that represent the exact normative state at each time step. By updating the contract state one step at a time, we ensure that the violation score for any given event is calculated strictly against the specific literals in force at that moment, accounting for all prior satisfactions, discharges, or triggered reparations. Then, propagates the updated residual to the subsequent evaluation step. For each event on a trace, we define a way to evaluate only the enforced



Month i
1
2
Event at i A_i
$\{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}$
$\{\mathsf{OCC}^{(1)}\}$
$\mathsf{Prog}(\pi'_{i-1}, C)$
$\lceil \mathsf{Notif\_T} \mid \Gamma^{+} \cdot \mathsf{Notif\_T} \rceil$ $\mathbf{Rep}(C_3)$
$\lceil \mathsf{Notif\_T} \mid \Gamma^{+} \cdot \mathsf{Notif\_T} \rceil$ $(\mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3))$

Figure 19: Progression on a guarded contract where the guard is not satisfied, and the inner contract triggers a reparation.

literal instead of the whole contract as an optimization step. To do so, we define the function literal head.

Definition 42 (Literal Head). *Let ε_ℓ be a distinguished symbol denoting the absence of an immediately available literal. The* literal head *function*

$$\mathsf{LH} : \mathsf{TACNL} \to (\ell \cup \{\varepsilon_\ell\}),$$

is defined inductively on the structure of contracts as follows:

$$\begin{aligned}
\mathsf{LH}(\ell) &:= \ell \\
\mathsf{LH}(C; C') &:= \mathsf{LH}(C) \\
\mathsf{LH}(C \blacktriangleright C') &:= \mathsf{LH}(C) \\
\mathsf{LH}(\langle\langle re \rangle\rangle C) &:= \begin{cases} \varepsilon_\ell, & \textit{if } re \neq \varepsilon, \\ \mathsf{LH}(C), & \textit{if } re = \varepsilon. \end{cases} \\
\mathsf{LH}(\lceil re \rceil C) &:= \begin{cases} \mathsf{LH}(C), & \textit{if } re \neq \varepsilon, \\ \varepsilon_\ell, & \textit{if } re = \varepsilon. \end{cases} \\
\mathsf{LH}(C^n) &:= \mathsf{LH}(C) \\
\mathsf{LH}(\mathbf{Rep}(C)) &:= \mathsf{LH}(C).
\end{aligned}$$

We move now to define the quantitative violation semantics based on: 1. the progress monitor returning which are the remaining parts of a contract that are still to be evaluated, 2. Scoring each event against the head literal of the residual contract returned by the Prog function. The final score of a function is thus the sum of violation for each event.

Definition 43 (Quantitative Violation Semantics). *Let $\langle A \rangle$ be a single event trace over Γ, π be a (possibly empty) finite trace Γ^*, and C be a contract from* TACNL. *We define* *the* quantitative violation semantics, *denoted by* $[\![]\!]_{qviol} : \Gamma^*, \mathsf{TACNL} \to \mathbb{N}$*, which maps a trace and a contract to a natural number representing the violation score of trace on the contract. The function is defined recursively by evaluating the head of the trace ($\langle A \rangle$) and propagating the residual contract to the tail (π):*

$$[\![\langle A\rangle \circ \pi, C]\!]_{qviol} := \begin{cases} [\![\langle A\rangle, C]\!]_{qviol} & \textit{if } \pi = \varepsilon \vee \mathsf{Prog}(\langle A\rangle, C) = \varepsilon_C, \\ [\![\langle A\rangle, C]\!]_{qviol} + [\![\pi, \mathsf{Prog}(\langle A\rangle, C)]\!]_{qviol} & \textit{otherwise}. \end{cases}$$

where the instantaneous violation score *for a single event* $\langle A\rangle$ *against a contract C is defined inductively on the structure of the contract:*

$$[\![\langle A\rangle, C]\!]_{qviol} := \begin{cases} [\![\langle A\rangle, C_1]\!]_{qviol} + [\![\langle A\rangle, C_2]\!]_{qviol} & \textit{if } C = C_1 \wedge C_2, \\ 0 & \textit{if } \neg(\langle A\rangle \models_{\perp'} \mathsf{LH}(C)) \textit{ or } \mathsf{LH}(C) = \varepsilon_\ell \\ 1 & \textit{if } \langle A\rangle \models_{\perp'} \mathsf{LH}(C), \end{cases}$$

Intuitively, the formula $[\![\langle A\rangle \circ \pi, C]\!]_{\text{qviol}}$ treats the contract execution as a path-accumulation problem. At every time step, the function:

1. **Snapshots the penalty:** It calculates $[\![\langle A\rangle, C]\!]_{\text{qviol}}$, which asks "Given the current literals fromC, does the current event A violate any of them?" This is a stateless check based purely on the structure of C at that instant.

2. **Updates the state:** It computes $\mathsf{Prog}(\langle A\rangle, C)$, effectively moving the contract pointer forward (e.g., from a paid obligation to the next month's rent, or from a violated duty to a reparation).

3. **Accumulates:** It adds the snapshot penalty to the result of the recursive call on the remaining trace using the *new* state.

Intuitively, the formula $[\![\langle A\rangle \circ \pi, C]\!]_{\text{qviol}}$ treats the contract execution as a path-accumulation problem. At every time step, the function:

1. **Snapshots the penalty:** It calculates $[\![\langle A\rangle, C]\!]_{\text{qviol}}$, which asks "Given the current literals from C, does the current event A violate any of them?" This is a stateless check based purely on the structure of C at that instant.

2. **Updates the state:** It computes $\mathsf{Prog}(\langle A\rangle, C)$, effectively moving the contract pointer forward (e.g., from a paid obligation to the next month's rent, or from a violated duty to a reparation).

3. **Accumulates:** It adds the snapshot penalty to the result of the recursive call on the remaining trace using the *new* state.

The explicit handling of Sequence and Reparation is done in the contract progress function, which ensures that "zero-delay" transitions are penalized correctly. For instance, if a contract requires C_1 then C_2, and an event discharges C_1 but violates C_2 in the same step, the summation logic ($[\![\langle A\rangle, C_1]\!]_{\text{qviol}} + [\![\langle A\rangle, C_2]\!]_{\text{qviol}}$) ensures the violation of C_2 is not ignored simply because it appeared in a continuation.

The definition of the instantaneous score $[\![\langle A\rangle, C]\!]_{\text{qviol}}$ rests on distinguishing between **concurrent** (parallel) obligations and **structural** (atomic) constraints. It decouples the measurement of the "volume" of non-compliance from the binary verification of specific rules.

Additivity of Concurrency. The case $[\![\langle A\rangle, C_1 \wedge C_2]\!]_{\text{qviol}} := [\![\langle A\rangle, C_1]\!]_{\text{qviol}} + [\![\langle A\rangle, C_2]\!]_{\text{qviol}}$ captures the "width" of the violation as show in Example.13. In normative systems, a conjunction represents distinct, independent obligations that are active simultaneously. By summing the scores, the function ensures that the penalty is proportional to the number of distinct parallel duties neglected in a single instant, preventing "violation masking" in which a single boolean verdict could hide multiple breaches.

Binary Structural Verdict. For constructs that are not distinct parallel duties (such as literals, sequences, or reparations), the definition relies on the binary tight violation relation ($\models_{\perp^t}$). This captures the "existence" of a fault in a non-decomposable structure. For example, a single atomic duty ($\mathbf{O}(a)$) can only be violated once per step.

Separation of State and Score. This approach assumes that the complexity of temporal evolution is handled by Prog, while $[\![]\!]_{\text{qviol}}$ handles the instantaneous cost. By reducing non-conjunction cases to a simple check ($\langle A\rangle \models_{\perp^t} C$), the definition asserts that scoring is local (checking if the current active node is broken), while progression is temporal (handling the flow from one obligation to the next).

We summarize these properties in the following theorem, which establishes that the quantitative score is a monotonically increasing function that acts as a "super-set" of the binary violation semantics. While a binary trace might be "Satisfied" (via reparation), the quantitative score reveals the cost of that path.

Theorem 7.1 (Consistency of Quantitative and Tight Semantics)**.** *For any contract C and finite trace π:*

1. ***Zero Score Implications:*** *If $[\![\pi, C]\!]_{qviol} = 0$, then exactly one of the following three disjoint cases holds:*
   - *(a) $\pi \models_{\top^t} C$ if and only if $\mathsf{Prog}\pi, C = \varepsilon_C$ and $\forall k < |\pi| - 1 : \mathsf{Prog}(\pi[0,k], C) \neq \varepsilon_C$.*
   - *(b) $\pi \models_{\top^p} C$ if and only if $\mathsf{Prog}\pi, C = \varepsilon_C$ and $\exists k < |\pi| - 1$ such that $\mathsf{Prog}(\pi[0,k], C) = \varepsilon_C$.*
   - *(c) $\pi \models_{?} C$ if and only if $\mathsf{Prog}\pi, C \neq \varepsilon_C$.*
2. ***Non-Zero Score Implications:*** *If $[\![\pi, C]\!]_{qviol} \neq 0$, then:*
   - *(a) $\pi \models_{\perp^t} C$ if and only if $[\![\pi, C]\!]_{qviol} = 1$ and $[\![\pi[0, |\pi| - 2], C]\!]_{qviol} = 0$.*
   - *(b) $\pi \models_{\perp^p} C$ if and only if $[\![\pi, C]\!]_{qviol} > 1$ or ($[\![\pi, C]\!]_{qviol} = 1$ and $[\![\pi[0, |\pi| - 2], C]\!]_{qviol} = 1$).*
3. ***Reparation Cost:*** *If π satisfies C strictly through a reparation mechanism (i.e., $\pi \models_{\top^t} C$ but primary obligations failed), then $[\![\pi, C]\!]_{qviol} > 0$.*

Proof. We prove the implications by structural induction on the trace π and the contract C, utilizing the definitions of the quantitative function $[\![]\!]_{\text{qviol}}$ and the contract progression Prog.

1. Zero Score Implications ($[\![\pi, C]\!]_{\mathbf{qviol}} = 0$) Assume $[\![\pi, C]\!]_{\text{qviol}} = 0$. By the definition of the cumulative score, this implies that for all steps $i < |\pi|$, the instantaneous penalty is zero: $[\![\langle A_i \rangle, \mathsf{Prog}(\pi_i, C)]\!]_{\text{qviol}} = 0$. Consequently, no tight violation has occurred at any step. The contract state evolves purely via Prog without triggering any penalty clauses.

1. **Case 1(a): Tight Satisfaction ($\models_{\top^t}$).**
   - ($\Rightarrow$) Assume $\mathsf{Prog}(\pi, C) = \varepsilon_C$ and for all strict prefixes π', $\mathsf{Prog}(\pi', C) \neq \varepsilon_C$. The condition $\mathsf{Prog}(\pi, C) = \varepsilon_C$ indicates that the contract has been fully discharged. Since the score is 0, this discharge was not achieved via a violation-triggered path (e.g., a reparation where the primary failed). The absence of ε_C in prior prefixes ensures that this is the *first* moment of discharge. By Definition 24 (Tight Satisfaction), the first prefix to fully satisfy the obligations corresponds to $\models_{\top^t}$.
   - ($\Leftarrow$) If $\pi \models_{\top^t} C$, then by Lemma **??** (Satisfaction Saturation), the progression must reach ε_C exactly at π. Since it is a *tight* satisfaction, no proper prefix could have satisfied it (reached ε_C) earlier.
2. **Case 1(b): Post Satisfaction ($\models_{\top^p}$).**
   - The condition $\exists k < |\pi|$ such that $\mathsf{Prog}(\pi[0,k], C) = \varepsilon_C$ implies that the contract was already discharged at a previous step k.
   - By Definition 22, $\pi \models_{\top^p} C$ holds if there exists a strict prefix that tightly satisfies C. Since the score is 0, the path to k was compliant. Thus, the state remains ε_C for the remainder of the trace, maintaining the $\models_{\top^p}$ status.
3. **Case 1(c): Pre Satisfaction ($\models_{?}$).**
   - Assume $\mathsf{Prog}(\pi, C) \neq \varepsilon_C$. Since $[\![\pi, C]\!]_{\text{qviol}} = 0$, no violation has occurred. However, the contract has not reduced to the empty contract ε_C, meaning active obligations remain.
   - This satisfies the definition of $\models_{?}$: the trace is neither satisfied ($\models_{\top^t}$ / $\models_{\top^p}$) nor violated ($\models_{\bot^t}$ / $\models_{\bot^p}$). It is effectively "pending."

2. Non-Zero Score Implications ($[\![\pi, C]\!]_{\mathbf{qviol}} \neq 0$) Assume $[\![\pi, C]\!]_{\text{qviol}} > 0$. This implies $\exists i$ such that $[\![\langle A_i \rangle, C_i]\!]_{\text{qviol}} > 0$.

1. **Case 2(a): Tight Violation ($\models_{\bot^t}$).**
   - We consider the case where $[\![\pi, C]\!]_{\text{qviol}} = 1$, the score of the immediate prefix is 0 and let $n = |\pi|$.

- $[\![\pi[0,n-2],C]\!]_{\text{qviol}} = 0$ implies that for all previous steps, the contract was in a compliant state ($\models_?$).
- The jump to $[\![\pi,C]\!]_{\text{qviol}} = 1$ implies that the instantaneous score at the last step $[\![\langle A_{n-1}\rangle, \mathsf{Prog}(\pi[0,n-2],C)]\!]_{\text{qviol}} = 1$.
- A positive instantaneous score corresponds to a tight violation of the active residual contract.
- Since this is the *first* non-zero score, it corresponds to the *first* prefix that triggers a violation. This corresponds to the definition of $\pi \models_{\perp^t} C$.

2. **Case 2(b): Post Violation ($\models_{\perp^p}$).**

   - The condition $[\![\pi,C]\!]_{\text{qviol}} > 1$ or ($[\![\pi,C]\!]_{\text{qviol}} = 1$ and $[\![prefix]\!]_{\text{qviol}} = 1$) implies that the violation score did not originate purely at the current step (or if it did, it was cumulative).
   - Specifically, if $[\![\pi[0,n-2],C]\!]_{\text{qviol}} \geq 1$, then a violation occurred strictly in the past.
   - By Definition 22, if a strict prefix tightly violated the contract ($\models_{\perp^t}$), the current trace is in $\models_{\perp^p}$. The non-zero score is carried forward monotonically.

3. **Case 2(c): Reparation Cost.**

   - Consider a contract $C_{primary} \blacktriangleright C_{repair}$.
   - If π satisfies this strictly through the reparation mechanism, it means π did *not* satisfy $C_{primary}$.
   - By the definition of reparation progression, the transition to C_{repair} occurs only if $\pi \models_{\perp^t} C_{primary}$.
   - By the definition of the instantaneous scoring function for reparation, $[\![\langle A\rangle, C_{primary} \blacktriangleright C_{repair}]\!]_{\text{qviol}} = 1 + \dots$ when the primary violates.

   Therefore, the path involving the repair accumulates a score of at least 1 (the penalty for breaking the primary), confirming $[\![\pi,C]\!]_{\text{qviol}} > 0$.

□

This theorem highlights the utility of the quantitative approach for post-hoc analysis: distinguishing between a "perfect" execution (Score 0) and a "compliant but costly" execution (Score > 0, e.g., paying fines), a distinction lost in the binary $\models_{\top^t}$ verdict.

To illustrate the lemma and the violation semantics, we study several examples:

Example 34 (Reparation: Tight vs. Quantitative Evaluation (Extension of Example 31 and Figures 15–16))**.** *We revisit Example 31 and explicitly evaluate both the forward-looking tight semantics and the quantitative violation semantics at each step.*

Let

$$C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \ \blacktriangleright\ \mathbf{O}_1(\mathsf{PAY\_F}).$$

Trace $\pi = \langle A_1 \rangle$ with $A_1 = \{\mathsf{PAY\_R}^{(12)}\}$.

- ***Progress:*** $\mathsf{Prog}(\langle A_1 \rangle, C_3) = \varepsilon_C$.
- ***Tight semantics:*** $\langle A_1 \rangle \models_{\top^t} C_3$.
- ***Quantitative score:*** $[\![\langle A_1 \rangle, C_3]\!]_{qviol} = 0$.

This illustrates immediate satisfaction with zero cost. Figure 20 summarizes the point-wise valuation for π.

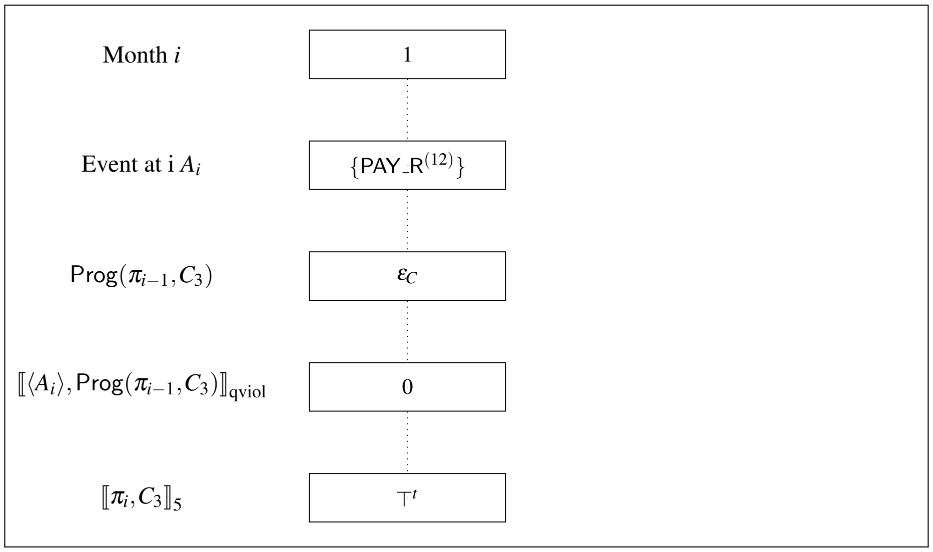

Month i
1
Event at i A_i
$\{\mathsf{PAY\_R}^{(12)}\}$
$\mathsf{Prog}(\pi_{i-1}, C_3)$
ε_C
$[\![\langle A_i \rangle, \mathsf{Prog}(\pi_{i-1}, C_3)]\!]_{\mathrm{qviol}}$
0
$[\![\pi_i, C_3]\!]_5$
$\top^t$

Figure 20: Pointwise valuation of the progress (Prog), violation score ($[\![]\!]_{\mathrm{qviol}}$), and tight satisfaction ($[\![]\!]_5$) for the contract $C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ under the trace $\pi = \langle A_1 \rangle$.

Trace $\pi' = \langle A'_1, A'_2 \rangle$ with $A'_1 = \{\mathsf{OCC}^{(1)}\}$ and $A'_2 = \{\mathsf{PAY\_F}^{(12)}\}$.

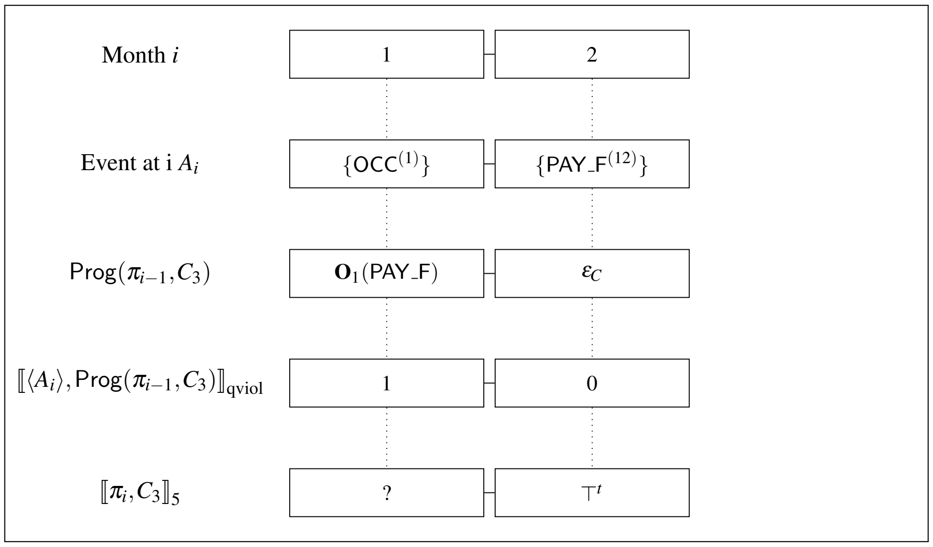
- ***Step 1:*** $\langle A'_1 \rangle \models_{\bot^t} \mathbf{O}_1(\mathsf{PAY\_R})$, *hence* $\mathsf{Prog}(\langle A'_1 \rangle, C_3) = \mathbf{O}_1(\mathsf{PAY\_F})$, *and* $[\![\langle A'_1 \rangle, C_3]\!]_{qviol} = 1$.
- ***Step 2:*** $\langle A'_2 \rangle \models_{\top^t} \mathbf{O}_1(\mathsf{PAY\_F})$, $\mathsf{Prog}(\pi', C_3) = \varepsilon_C$, *and* $[\![\langle A'_2 \rangle, \mathbf{O}_1(\mathsf{PAY\_F})]\!]_{qviol} = 0$.

Overall, $\pi' \models_{\top^t} C_3$ but $[\![\pi', C_3]\!]_{qviol} = 1$, illustrating Theorem 7.1(3). Figure 21 summarizes the point-wise valuation for π'.

Example 35 (Infinite Repetition with Accumulated Cost (Extension of Example 17))**.** *We extend Example (Progression of Infinite Repetition) for*

$$\mathbf{Rep}(C_3) = \mathbf{Rep}(\mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})).$$

Consider the trace $\pi = \langle A_1, A_2 \rangle$ with $A_1 = \{\mathsf{PAY\_R}^{(12)}\}$ and $A_2 = \{\mathsf{OCC}^{(1)}\}$.



Month i
1
2
Event at i A_i
$\{\mathsf{OCC}^{(1)}\}$
$\{\mathsf{PAY\_F}^{(12)}\}$
$\mathsf{Prog}(\pi_{i-1}, C_3)$
$\mathbf{O}_1(\mathsf{PAY\_F})$
ε_C
$[\![\langle A_i\rangle, \mathsf{Prog}(\pi_{i-1}, C_3)]\!]_{\text{qviol}}$
1
0
$[\![\pi_i, C_3]\!]_5$
?
$\top^t$

Figure 21: Pointwise valuation of the progress (Prog), violation score ($[\![]\!]_{\text{qviol}}$), and tight satisfaction ($[\![]\!]_5$) for the contract $C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$ under the trace $\pi' = \langle A'_1, A'_2\rangle$.

- ***Step 1:*** $\langle A_1\rangle \models_{\top^t} C_3$*,* $\mathsf{Prog}(\langle A_1\rangle, \mathbf{Rep}(C_3)) = \mathbf{Rep}(C_3)$*, and* $[\![\langle A_1\rangle, \mathbf{Rep}(C_3)]\!]_{qviol} = 0$.
- ***Step 2:*** $\langle A_2\rangle \models_{\perp^t} C_3$*,* $\mathsf{Prog}(\langle A_2\rangle, \mathbf{Rep}(C_3)) = \mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3)$*, and* $[\![\langle A_2\rangle, \mathbf{Rep}(C_3)]\!]_{qviol} = 1$.

Thus, $\pi \models_{\perp^p} \mathbf{Rep}(C_3)$ *while* $[\![\pi, \mathbf{Rep}(C_3)]\!]_{qviol} = 1$*. A longer trace that later satisfies the fine would increase satisfaction without decreasing the accumulated score, illustrating monotonicity. Figure 22 summarizes the point-wise valuation for this trace.*

Example 36 (Guarded Contract: Divergence Between Status and Cost (Extension of Example 18–19))**.** *We extend Example (Progression of Guarded Contracts) for*

$$C := \lceil \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)} \rceil \mathbf{Rep}(C_3).$$

Trace $\pi = \langle A_1, A_2\rangle$ **with** $A_1 = \{\mathsf{PAY\_R}^{(12)}\}$ **and** $A_2 = \{\mathsf{Notif\_T}^{(1)}\}$**.**

- ***Step 1:*** $\langle A_1\rangle \models_{?} C$*,* $\mathsf{Prog}(\langle A_1\rangle, C) = C$*, and* $[\![\langle A_1\rangle, C]\!]_{qviol} = 0$.
- ***Step 2:*** $\langle A_2\rangle \models_{\top^t} C$*,* $\mathsf{Prog}(\pi, C) = \varepsilon_C$*, and* $[\![\langle A_2\rangle, C]\!]_{qviol} = 0$.

Hence, $\pi \models_{\top^t} C$ *and* $[\![\pi, C]\!]_{qviol} = 0$*. Figure 23 summarizes the point-wise valuation for* π*.*

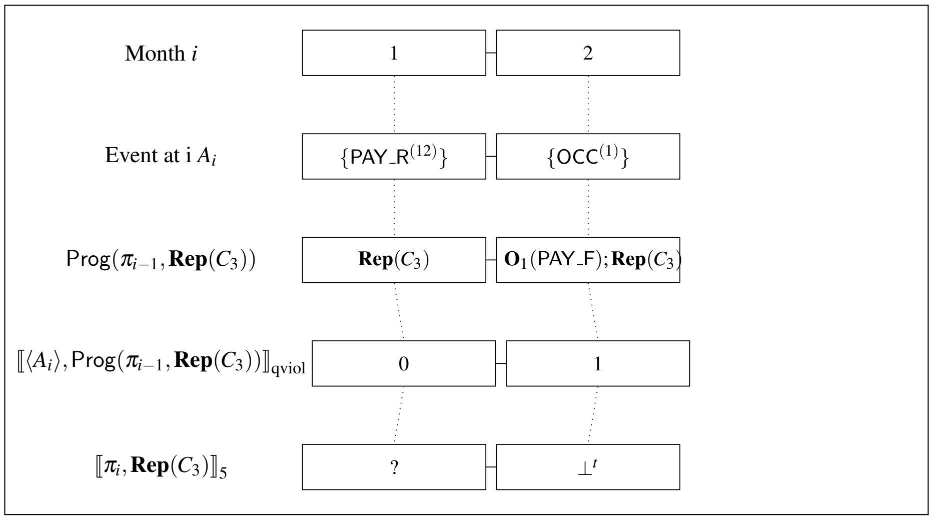

Month i
1
2
Event at i A_i
$\{\mathsf{PAY\_R}^{(12)}\}$
$\{\mathsf{OCC}^{(1)}\}$
$\mathsf{Prog}(\pi_{i-1}, \mathbf{Rep}(C_3))$
$\mathbf{Rep}(C_3)$
$\mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3)$
$[\![\langle A_i \rangle, \mathsf{Prog}(\pi_{i-1}, \mathbf{Rep}(C_3))]\!]_{\text{qviol}}$
0
1
$[\![\pi_i, \mathbf{Rep}(C_3)]\!]_5$
?
$\bot^t$

Figure 22: Pointwise valuation of the progress (Prog), violation score ($[\![]\!]_{\text{qviol}}$), and tight satisfaction ($[\![]\!]_5$) for the contract $\mathbf{Rep}(C_3)$ under the trace $\pi = \langle A_1, A_2 \rangle$.

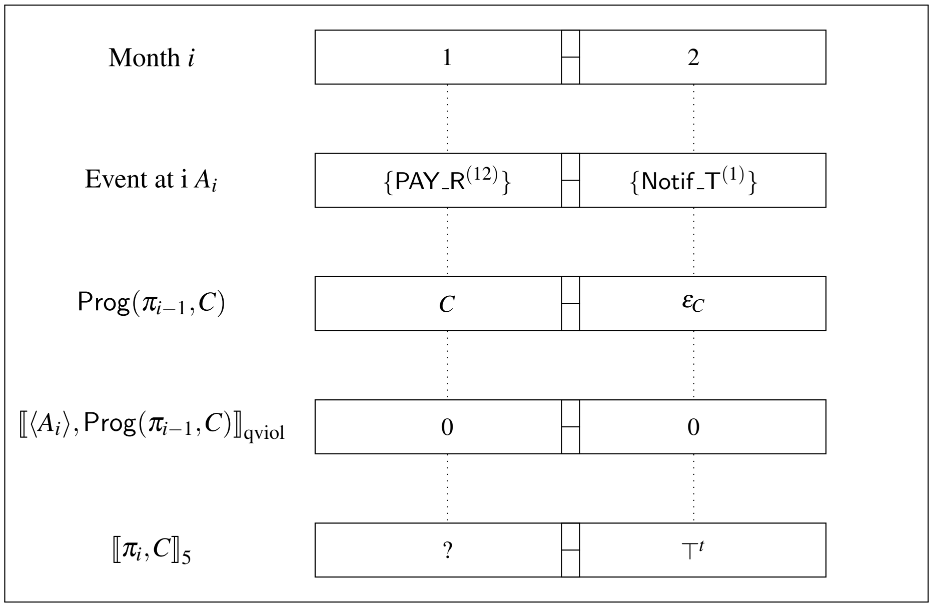
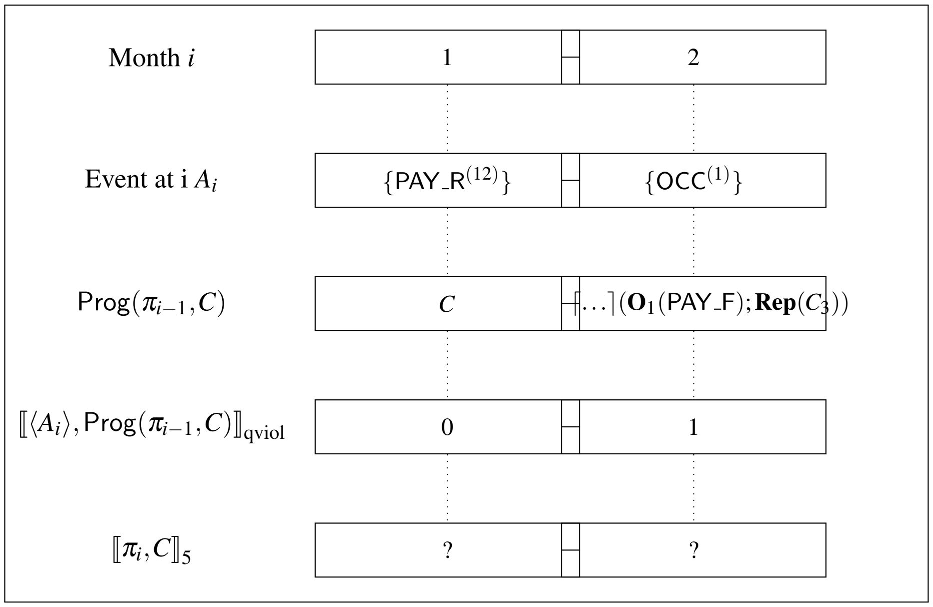

Month i
1
2
Event at i A_i
$\{\mathsf{PAY\_R}^{(12)}\}$
$\{\mathsf{Notif\_T}^{(1)}\}$
$\mathsf{Prog}(\pi_{i-1}, C)$
C
ε_C
$[\![\langle A_i \rangle, \mathsf{Prog}(\pi_{i-1}, C)]\!]_{\text{qviol}}$
0
0
$[\![\pi_i, C]\!]_5$
?
$\top^t$

Figure 23: Pointwise valuation of the progress (Prog), violation score ($[\![]\!]_{\text{qviol}}$), and tight satisfaction ($[\![]\!]_5$) for the contract $C := \lceil \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)} \rceil \mathbf{Rep}(C_3)$ under the trace $\pi = \langle A_1, A_2 \rangle$.

Trace $\pi' = \langle A_1, A_2' \rangle$ with $A_2' = \{\mathsf{OCC}^{(1)}\}$.

- ***Step 2:*** *$\langle A_2' \rangle \models_{\perp^t} \mathbf{Rep}(C_3)$, $\mathsf{Prog}(\pi', C) = \lceil \ldots \rceil (\mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3))$, and $[\![\langle A_2' \rangle, C]\!]_{qviol} =$ 1.*

Thus, $\pi' \models_{?} C$ in the tight semantics but $[\![\pi', C]\!]_{qviol} = 1$, directly illustrating that quantitative semantics exposes violations masked by guards and reparations. Figure 24 summarizes the pointwise valuation for π'.



Month i	1	2
Event at i A_i	$\{\mathsf{PAY\_R}^{(12)}\}$	$\{\mathsf{OCC}^{(1)}\}$
$\mathsf{Prog}(\pi_{i-1}, C)$	C	$\lceil \ldots \rceil (\mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3))$
$[\![\langle A_i \rangle, \mathsf{Prog}(\pi_{i-1}, C)]\!]_{\mathrm{qviol}}$	0	1
$[\![\pi_i, C]\!]_5$	?	?

Figure 24: Pointwise valuation of the progress (Prog), violation score ($[\![]\!]_{\mathrm{qviol}}$), and tight satisfaction ($[\![]\!]_5$) for the contract $C := \lceil \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)} \rceil \mathbf{Rep}(C_3)$ under the trace $\pi' = \langle A_1, A_2' \rangle$.

Conclusion. This section introduced the quantitative violation semantics $[\![]\!]_{\mathrm{qviol}}$ as a trace-level cost measure that complements the tight, three-valued contract semantics. By coupling a local, instantaneous penalty with the residual evolution induced by the progress function Prog, $[\![]\!]_{\mathrm{qviol}}$ cleanly separates temporal state change from scoring, while remaining sensitive to reparations, sequencing, and other structural constructs. Theorem 7.1 makes this link precise: a zero score characterizes traces that are fully compliant (either tightly satisfied, post-satisfied, or still pending), whereas any positive score pinpoints the presence and persistence of violations, including those that are later repaired and therefore masked at the level of a binary satisfaction verdict. The subsequent examples validate this intuition operationally by tracking Prog, $[\![]\!]_{\mathrm{qviol}}$, and $[\![]\!]_5$ pointwise, showing how the quantitative view supports post hoc audit, comparison of alternative executions, and downstream optimization tasks where "satisfied" is not a sufficient notion of quality.

7.4 Quantitative Violation Monitor Construction

While the forward-looking semantics stops at the first decisive violation, the quantitative semantics requires a monitor that persists on the extended run after the violation, whilst continuing to accumulate violation points over time. To avoid the complexities of using counting machines, we define the *Quantitative Monitor* as a Moore machine whose output alphabet is the set of natural numbers: each state is associated with an instantaneous violation score. Consequently, the overall score for a given trace is derived from the cumulative sum of the machine's outputs at each execution step. Furthermore, this position-based scoring enables precise localization of the time points at which violations occur, thereby enhancing explainability.

Definition 44 (Quantitative Monitor)**.** *The* Quantitative Monitor*, written $\mathcal{M}^{qt}$, is a Mealy machine whose output alphabet is elements from $\mathbb{N}$ representing a score.*

$$\mathcal{M}^{qt} = (Q, q_0, \Gamma, \delta, \lambda_{\mathbb{N}}),$$

where:

1. *Q is the set of states.*
2. *$q_0 \in Q$ is the initial state.*
3. *Γ is the input event alphabet.*
4. *$\lambda = Q \times \Gamma \times \to \mathbb{N}$ is scoring function.*
5. *$\delta : Q \times \Gamma \to Q$ is the transition function.*

Definition 45 (Quantitative Execution Score)**.** *Let $\mathcal{M}^{qt} = (Q, q_0, \Gamma, \delta, \lambda)$ be a Quantitative Monitor and let $\pi = \langle A_0, A_1, \dots, A_{n-1} \rangle \in \Gamma^*$ be a finite trace. The execution of $\mathcal{M}^{qt}$ on π produces an execution $\langle q_0, A_0, q_1, A_1 \dots, A_{n-1}, q_n \rangle$ such that $q_{i+1} = \delta(q_i, A_i)$ for all $0 \leq i < n$.*

The quantitative score *of the trace π on $\mathcal{M}^{qt}$, denoted as $\mathsf{Score}(\mathcal{M}^{qt}, \pi)$, is defined as the sum of the instantaneous scores of the states visited during the run, excluding the initial state:*

$$\mathsf{Score}(\mathcal{M}^{qt}, \pi) = \sum_{i=0}^{|\pi|-1} \lambda(q_i, \pi(i)).$$

To make the monitor construction constructive and finite (where possible), we define it inductively. However, since the states are residual contracts, the primary challenge is efficiently handling *conjunction*.

Definition 46 (Quantitative Monitor Construction)**.** *The* Quantitative Monitor Construction *function that returns for a contract $C \in \mathsf{TACNL} : (Q, q_0, \Gamma, \delta, \lambda)$ and returns a Quantitative Monitor $\mathcal{M}^{qt}$:=, denoted $\mathsf{QMC}(C)$, builds the automaton for contract C is defined as :*

1. *$Q \subset \mathsf{TACNL} \times \varepsilon_C$.*
2. *$q_0 = C$.*

3. $\delta(q_i, A) := \mathsf{Prog}(\langle A\rangle, q_i)$.

4. $\lambda(q_i, A) := [\![\langle A\rangle, q_i]\!]_{qviol}$.

Example 37 (Quantitative Monitor for a Guarded Repetition)**.** *We construct (a finite fragment of) the quantitative monitor* $\mathsf{QMC}(C)$ *for the guarded contract*

$$C := \lceil \Gamma^{+} \cdot \mathsf{Notif\_T}^{(1)} \rceil \mathbf{Rep}(C_3), \qquad \textit{where } C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \ \blacktriangleright\ \mathbf{O}_1(\mathsf{PAY\_F}).$$

Let $re := \Gamma^{+} \cdot \mathsf{Notif\_T}^{(1)}$. *We write* $\lceil re \rceil \mathbf{Rep}(C_3)$ *for the guard instance used below. Starting from the contract as the initial state:*

$$q_0 := \lceil \Gamma^{+} \cdot \mathsf{Notif\_T}^{(1)} \rceil \mathbf{Rep}(C_3),$$

The states of the Mealy machine are obtained by exhaustively analyzing which classes of events $A \in \Gamma$ *produce a change in the residual via the progress function* $\mathsf{Prog}(\langle A\rangle, q)$, *and by recording the corresponding instantaneous cost* $[\![\langle A\rangle, q]\!]_{qviol}$.

From q_0*, only the satisfaction or violation of the head obligation of* C_3 *can affect progression, since the guard does not discharge until a termination notification occurs. The head of the repetition body is the primary obligation* $\mathbf{O}_1(\mathsf{PAY\_R})$. *Therefore, two event classes are relevant.*

If $A \in \mathsf{PAY\_R}^{(12)}$*, the primary obligation is satisfied. No violation is incurred, hence*

$$[\![\langle A\rangle, q_0]\!]_{qviol} = 0,$$

and progression consumes the obligation and re-enters the repetition:

$$\mathsf{Prog}(\langle A\rangle, q_0) = \mathbf{Rep}(C_3).$$

This residual is represented as state q_2.

If $A \in \overline{\mathsf{PAY\_R}^{(12)}}$*, the primary obligation is violated. This yields a unit penalty*

$$[\![\langle A\rangle, q_0]\!]_{qviol} = 1,$$

and progression activates the reparation clause, producing the residual

$$\mathsf{Prog}(\langle A\rangle, q_0) = \mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3),$$

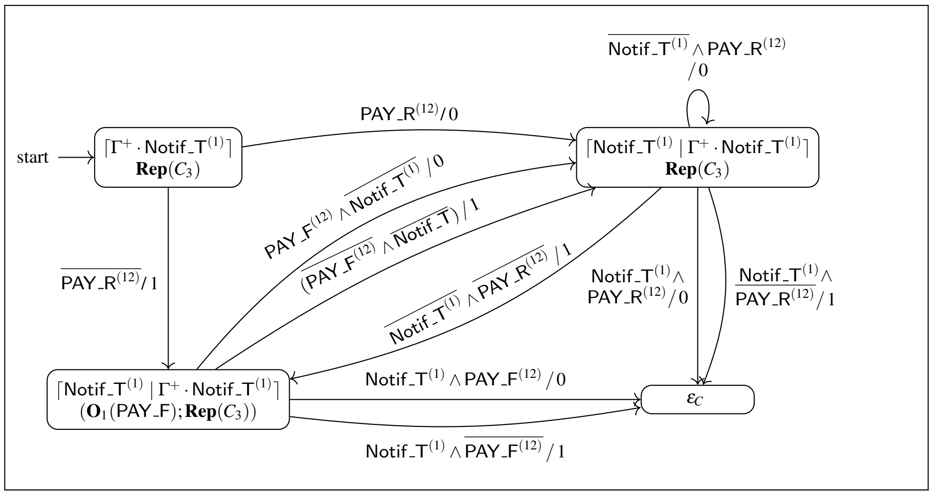
which is represented as state q_1.

From q_1*, the head obligation is now the fine obligation* $\mathbf{O}_1(\mathsf{PAY\_F})$. *Again, only events that affect its satisfaction or the guard condition are relevant. If* $A \in \mathsf{PAY\_F}^{(12)}$ *and no termination notification occurs, the fine is satisfied with zero cost and progression returns to the repetition residual* q_2. *If* $A \in \overline{\mathsf{PAY\_F}^{(12)}}$ *and no termination occurs, the fine is violated, yielding cost* 1*, but progression still returns to the repetition, as the fine obligation is consumed after one step.*

If a termination notification $A \in \mathsf{Notif\_T}^{(1)}$ *occurs in state* q_1*, the guard condition is fulfilled. Progression, therefore, yields the empty contract* ε_C*, independently of whether the fine is satisfied or violated, while the instantaneous cost is determined by the compliance of the active head obligation at that step.*

From q_2, the repetition phase is stable. Successful payment events $A \in \mathsf{PAY\_R}^{(12)}$ yield zero cost and leave the residual unchanged, resulting in a self-loop. Failed payment events incur a unit penalty and return progress to the fine residual q_1. As before, the occurrence of a termination notification immediately discharges the contract to ε_C, with the instantaneous cost reflecting whether the payment obligation was satisfied at that final step.

Collecting all distinct residuals reachable from q_0 under these event classes yields exactly the states and transitions shown in Figure 25.



start
$\lceil \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)} \rceil$ $\mathbf{Rep}(C_3)$
$\lceil \mathsf{Notif\_T}^{(1)} \mid \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)} \rceil$ $\mathbf{Rep}(C_3)$
$\lceil \mathsf{Notif\_T}^{(1)} \mid \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)} \rceil$ $(\mathbf{O}_1(\mathsf{PAY\_F}); \mathbf{Rep}(C_3))$
ε_C
$\overline{\mathsf{Notif\_T}^{(1)}} \wedge \mathsf{PAY\_R}^{(12)} / 0$
$\mathsf{PAY\_R}^{(12)} / 0$
$\overline{\mathsf{PAY\_R}^{(12)}} / 1$
$\mathsf{PAY\_F}^{(12)} \wedge \overline{\mathsf{Notif\_T}^{(1)}} / 0$
$(\overline{\mathsf{PAY\_F}^{(12)}} \wedge \overline{\mathsf{Notif\_T}}) / 1$
$\overline{\mathsf{Notif\_T}^{(1)}} \wedge \overline{\mathsf{PAY\_R}^{(12)}} / 1$
$\mathsf{Notif\_T}^{(1)} \wedge \mathsf{PAY\_R}^{(12)} / 0$
$\mathsf{Notif\_T}^{(1)} \wedge \overline{\mathsf{PAY\_R}^{(12)}} / 1$
$\mathsf{Notif\_T}^{(1)} \wedge \mathsf{PAY\_F}^{(12)} / 0$
$\mathsf{Notif\_T}^{(1)} \wedge \overline{\mathsf{PAY\_F}^{(12)}} / 1$

Figure 25: The quantitative violation monitor $\mathsf{QMC}(C)$ for $C := \lceil re \rceil \mathbf{Rep}(C_3)$, with $re := \Gamma^+ \cdot \mathsf{Notif\_T}^{(1)}$ and $C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$. On the transitions:
$\mathsf{PAY\_R}^{(12)} := \{A \in \Gamma \mid \{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\} \subseteq A\}$ a shorthand for when the payement successfully occured.
$\mathsf{Notif\_T}^{(1)} := \{A \in \Gamma \mid \{\mathsf{Notif\_T}^{(1)}\} \subseteq A\}$ for agent (1) sending a terminiation notification.
$\overline{P} := \{A \in \Gamma \setminus P\}$.
$P \wedge Q := \{A \mid A \in P \cap Q\}$.

We now compute the quantitative execution score for a trace in which the termination notification occurs at the third event. Let

$$\pi = \langle A_1, A_2, A_3 \rangle$$

with $A_1 := \{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\}$, $A_2 = \{\mathsf{OCC}^1\}$, *and* $A_3 := \{\mathsf{Notif\_T}^{(1)}, \mathsf{PAY\_F}^{(12)}\}$.

By inspection of Figure 25 and by Definition 45, the unique run of $\mathsf{QMC}(C)$ *on* π *is*

$$q_0 \xrightarrow{A_1/0} q_2 \xrightarrow{A_2/1} q_1 \xrightarrow{A_3/0} \varepsilon_C.$$

Therefore, the cumulative quantitative score is

$$(\mathsf{QMC(C)}, \pi) = \lambda(\mathsf{q_0}, \mathsf{A_1}) + \lambda(\mathsf{q_2}, \mathsf{A_2}) + \lambda(\mathsf{q_1}, \mathsf{A_3}) = 0 + 1 + 0 = 1.$$

This shows that although the trace satisfies the guarded contract in the tight semantics due to termination at the third step, the quantitative monitor records the intermediate violation as a persistent cost.

Theorem 7.2 (Correctness of Quantitative Monitor)**.** *Let C be a contract and $\pi = \langle A_0,\dots,A_n\rangle$ be a trace. Then the cumulative quantitative violation score is exactly the sum of the monitor outputs:*

$$[\![\pi, C]\!]_{qviol} = \mathsf{Score}(\mathsf{QMC}(C), \pi).$$

Proof. Let $\pi = \langle A_0,\dots,A_{n-1}\rangle$ be a finite trace of length n. Write $q_0 := C$ and, for each $0 \le i < n$, define the unique run of $\mathsf{QMC}(C)$ on π by

$$q_{i+1} := \delta(q_i, A_i) = \mathsf{Prog}(\langle A_i\rangle, q_i).$$

By definition of the monitor output function, for every i we have

$$\lambda(q_i, A_i) = [\![\langle A_i\rangle, q_i]\!]_{\text{qviol}}.$$

Hence, by Definition 45,

$$\mathsf{Score}(\mathsf{QMC}(C), \pi) = \sum_{i=0}^{n-1} \lambda(q_i, A_i) = \sum_{i=0}^{n-1} [\![\langle A_i\rangle, q_i]\!]_{\text{qviol}}.$$

It remains to show that $[\![\pi, C]\!]_{\text{qviol}}$ expands to the same sum. We prove by induction on n that

$$[\![\pi, C]\!]_{\text{qviol}} = \sum_{i=0}^{n-1} [\![\langle A_i\rangle, q_i]\!]_{\text{qviol}}.$$

Base case ($n = 1$). If $\pi = \langle A_0\rangle$, then by Definition (Quantitative Violation Semantics) $[\![\pi, C]\!]_{\text{qviol}} = [\![\langle A_0\rangle, C]\!]_{\text{qviol}} = [\![\langle A_0\rangle, q_0]\!]_{\text{qviol}}$. This is exactly the required sum.

Inductive step. Assume the claim holds for all traces of length n. Let $\pi' = \langle A_0,\dots,A_n\rangle$ be a trace of length $n+1$. Unfolding the recursive definition of $[\![]\!]_{\text{qviol}}$ yields

$$\begin{aligned}[\![\pi', C]\!]_{\text{qviol}} &= [\![\langle A_0\rangle, C]\!]_{\text{qviol}} + [\![\langle A_1,\dots,A_n\rangle, \mathsf{Prog}(\langle A_0\rangle, C)]\!]_{\text{qviol}}\\ &= [\![\langle A_0\rangle, q_0]\!]_{\text{qviol}} + [\![\langle A_1,\dots,A_n\rangle, q_1]\!]_{\text{qviol}}.\end{aligned}$$

Applying the induction hypothesis to the suffix trace $\langle A_1,\dots,A_n\rangle$ with initial contract q_1 gives

$$[\![\langle A_1,\dots,A_n\rangle, q_1]\!]_{\text{qviol}} = \sum_{i=1}^{n} [\![\langle A_i\rangle, q_i]\!]_{\text{qviol}}.$$

Therefore,

$$[\![\pi', C]\!]_{\text{qviol}} = [\![\langle A_0\rangle, q_0]\!]_{\text{qviol}} + \sum_{i=1}^{n} [\![\langle A_i\rangle, q_i]\!]_{\text{qviol}} = \sum_{i=0}^{n} [\![\langle A_i\rangle, q_i]\!]_{\text{qviol}}.$$

This establishes the inductive claim.

Combining the two displayed equalities, we obtain
$[\![\pi, C]\!]_{\text{qviol}} = \sum_{i=0}^{n-1} [\![\langle A_i\rangle, q_i]\!]_{\text{qviol}} = \mathsf{Score}(\mathsf{QMC}(C), \pi)$. □

Conclusion of the quantitative monitor construction. The monitor construction $\mathsf{QMC}(C)$ gives a finite, executable representation of the quantitative violation semantics whenever the set of reachable residual contracts under Prog is finite. Its key benefit is operational: instead of re-evaluating a contract from scratch on each prefix, we track the evolving residual contract as the current state, and we emit a local score via $\lambda(q,A) = [\![\langle A\rangle, q]\!]_{\mathrm{qviol}}$. The correctness theorem above guarantees that summing these transition-local outputs exactly recovers the trace-level cost $[\![\pi, C]\!]_{\mathrm{qviol}}$. In the next section, we refine this cost into an *attribution* measure by splitting the accumulated score across agents.

7.5 Quantitative Blame Semantics

While $[\![\pi, C]\!]_{\mathrm{qviol}}$ reports *how much* a trace deviates from a contract, it does not explain *who* caused the deviation in a multi-agent setting. Quantitative blame addresses this gap by refining each unit violation into an agent-indexed contribution. Concretely, we replace the scalar output alphabet $\mathbb{N}$ by vectors in $\mathbb{N}^2$, where the first component counts violations attributed to agent 1 and the second counts violations attributed to agent 2. The definition below corresponds to the structure of $[\![]\!]_{\mathrm{qviol}}$ and reuses the same progress-based unfolding of residual contracts.

Definition 47 (Quantitative Blame Semantics)**.** *Let $\pi \in \Gamma^*$ be a finite trace and $C \in \mathsf{TACNL}$ be a contract. The* quantitative blame semantics *is a function*

$$[\![\cdot]\!]_{qblame} : \Gamma^* \times \mathsf{TACNL} \to \mathbb{N}^2$$

that maps a trace and a contract to a blame vector (n_1, n_2), where n_i counts the number of violations attributed to agent i.

The function is defined recursively on the trace structure as follows:

$$[\![\langle A\rangle \circ \pi, C]\!]_{qblame} := \begin{cases} [\![\langle A\rangle, C]\!]_{qblame} & \text{if } \pi = \varepsilon \vee \mathsf{Prog}(\langle A\rangle, C) = \varepsilon_C, \\ [\![\langle A\rangle, C]\!]_{qblame} + [\![\pi, \mathsf{Prog}(\langle A\rangle, C)]\!]_{qblame} & \text{otherwise}. \end{cases}$$

The instantaneous quantitative blame *for a single event $\langle A\rangle$ against a contract C is defined by:*

$$[\![\langle A\rangle, C]\!]_{qblame} := \begin{cases} [\![\langle A\rangle, C_1]\!]_{qblame} + [\![\langle A\rangle, C_2]\!]_{qblame} & \text{if } C = C_1 \wedge C_2, \\ (1,0) & \text{if } \langle A\rangle \vDash_{\perp_1^t} \mathsf{LH}(C), \\ (0,1) & \text{if } \langle A\rangle \vDash_{\perp_2^t} \mathsf{LH}(C), \\ (0,0) & \text{otherwise}. \end{cases}$$

Additivity and accumulation. The recursive clause uses the usual component-wise addition on $\mathbb{N}^2$: if an event contributes (i, j) at the current step and the suffix contributes (k, l), then their sum is $(i+k, j+l)$. Hence, the blame vector accumulated over a trace counts, for each agent separately, how many step-local violations were attributed to that agent along the unique progress-induced run. This corresponds to the scalar accumulation in $[\![]\!]_{\mathrm{qviol}}$, but preserves per-agent accountability.

Why there is no $(1,1)$ case. We do not include a rule for a joint violation vector $(1,1)$ in the definition of $[\![\langle A\rangle, C]\!]_{\text{qblame}}$, such as a hypothetical condition $\langle A\rangle \vDash_{\perp^t_{12}} \mathsf{LH}(C)$. This is intentional: for the atomic head literal $\mathsf{LH}(C)$, the blame verdicts used in $\vDash_{\perp^t_i}$ are mutually exclusive in the underlying tight blame semantics, so a single literal cannot be violated by both agents in the same step. When multiple obligations are active concurrently, the conjunction case $C = C_1 \wedge C_2$ already captures multiple violations in one step by summing the vectors returned for each component, and this can yield a total vector whose two components are both positive, but only via *distinct* literals.

Example 38 (Blame Attribution and Quantitative Scores on the Same Traces)**.** *We reuse the contracts and traces from the quantitative violation semantics section and evaluate them simultaneously under the forward-looking blame semantics and the quantitative violation score.*

Recall the contract

$$C_3 := \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F}),$$

and the permission

$$C_2 := \mathbf{P}_1(\mathsf{OCC}).$$

Trace $\pi = \langle A_1, A_2\rangle$ **with** $A_1 = \{\mathsf{OCC}^{(1)}\}$ **and** $A_2 = \{\mathsf{PAY\_F}^{(12)}\}$**.** Step 1. *The active obligation is* $\mathbf{O}_1(\mathsf{PAY\_R})$*. Since* A_1 *contains no* $\mathsf{PAY\_R}^{(1)}$*, the obligation is violated. Under the blame semantics,*

$$\langle A_1\rangle \vDash_{\perp^t_1} \mathbf{O}_1(\mathsf{PAY\_R}),$$

so agent 1 is blamed. Quantitatively,

$$[\![\langle A_1\rangle, C_3]\!]_{qviol} = 1.$$

Progression activates the reparation:

$$\mathsf{Prog}(\langle A_1\rangle, C_3) = \mathbf{O}_1(\mathsf{PAY\_F}).$$

Step 2. *The active obligation is now* $\mathbf{O}_1(\mathsf{PAY\_F})$*. Since* $A_2 = \{\mathsf{PAY\_F}^{(12)}\}$*, the obligation is satisfied. No new blame arises, and no quantitative penalty is incurred:*

$$\langle A_2\rangle \nvDash_{\perp^t_S} \mathbf{O}_1(\mathsf{PAY\_F}) \quad \textit{for all } S, \qquad [\![\langle A_2\rangle, \mathbf{O}_1(\mathsf{PAY\_F})]\!]_{qviol} = 0.$$

Cumulative result. *The tight semantics yields* $\pi \models_{\top^t} C_3$*, while blame semantics records a single tight violation by agent 1. The quantitative score is obtained by summation:*

$$[\![\pi, C_3]\!]_{qviol} = [\![\langle A_1\rangle, C_3]\!]_{qviol} + [\![\langle A_2\rangle, \mathsf{Prog}(\langle A_1\rangle, C_3)]\!]_{qviol} = 1 + 0 = 1.$$

Trace $\pi' = \langle A_1 \rangle$ with $A_1 = \{\mathsf{OCC}^{(1)}\}$. *Consider now the conjunction $C_2 \wedge C_3$.*

The permission $\mathbf{P}_1(\mathsf{OCC})$ is violated because the subject attempts OCC without cooperation:

$$\langle A_1 \rangle \vDash_{\perp_2^t} \mathbf{P}_1(\mathsf{OCC}).$$

Simultaneously, the obligation $\mathbf{O}_1(\mathsf{PAY\_R})$ is violated, yielding

$$\langle A_1 \rangle \vDash_{\perp_1^t} \mathbf{O}_1(\mathsf{PAY\_R}).$$

Hence, blame semantics assigns joint responsibility:

$$\langle A_1 \rangle \vDash_{\perp_{12}^t} (C_2 \wedge C_3).$$

Quantitatively, both conjuncts contribute independently:

$$[\![\langle A_1 \rangle, C_2 \wedge C_3]\!]_{qviol} = [\![\langle A_1 \rangle, C_2]\!]_{qviol} + [\![\langle A_1 \rangle, C_3]\!]_{qviol} = 1 + 1 = 2.$$

Quantitative blame. *By Definition 47, the instantaneous blame at Step 1 is*

$$[\![\langle A_1 \rangle, C_3]\!]_{qblame} = (1, 0),$$

since $\langle A_1 \rangle \vDash_{\perp_1^t} \mathbf{O}_1(\mathsf{PAY\_R})$. At Step 2, no violation occurs, hence

$$[\![\langle A_2 \rangle, \mathbf{O}_1(\mathsf{PAY\_F})]\!]_{qblame} = (0, 0).$$

Therefore, the cumulative blame vector is obtained by component-wise addition:

$$[\![\pi, C_3]\!]_{qblame} = (1, 0) + (0, 0) = (1, 0).$$

So far, we have introduced two quantitative views of non-compliance. The first one, $[\![\pi, C]\!]_{\mathrm{qviol}}$, gives a single number that counts the number of violations along the trace, without caring who caused them. The second one, $[\![\pi, C]\!]_{\mathrm{qblame}}$, keeps the same idea but splits the count into two parts, one per agent. Since both definitions use the same progress function and add up step-by-step contributions, it is natural to ask whether the split version still matches the original violation semantics total. In other words, does adding up the two blame counters always give back the same score as $[\![\pi, C]\!]_{\mathrm{qviol}}$? The following lemma answers this by proving that the overall quantitative violation score is exactly the sum of the two blame components, so quantitative blame is just a finer view of the same quantity, not a different measure.

Lemma 18 (The relation of the blame score to the violation score)**.** *Let $\pi \in \Gamma^*$ be a finite trace and $C \in \mathsf{TACNL}$ be a contract.*

$$\textit{If } [\![\pi, C]\!]_{qblame} = (n_1, n_2)$$

is the quantitative blame vector for π and C, then the quantitative violation score decomposes as

$$[\![\pi, C]\!]_{qviol} = n_1 + n_2.$$

Proof. We prove the claim by induction on the length of the trace π.

Base case. Let $\pi = \langle A \rangle$ be a single-event trace. By Definitions 47 and (Quantitative Violation Semantics), both $[\![\langle A \rangle, C]\!]_{\text{qviol}}$ and $[\![\langle A \rangle, C]\!]_{\text{qblame}}$ are defined solely from the instantaneous evaluation of the head literal $\mathsf{LH}(C)$.

If $\langle A \rangle$ violates no active literal, then

$$[\![\langle A \rangle, C]\!]_{\text{qviol}} = 0 \quad \text{and} \quad [\![\langle A \rangle, C]\!]_{\text{qblame}} = (0,0).$$

If $\langle A \rangle$ violates $\mathsf{LH}(C)$ and the violation is attributed to agent i, then

$$[\![\langle A \rangle, C]\!]_{\text{qviol}} = 1 \quad \text{and} \quad [\![\langle A \rangle, C]\!]_{\text{qblame}} = \begin{cases} (1,0) & \text{if } i = 1, \\ (0,1) & \text{if } i = 2. \end{cases}$$

In all cases, $[\![\langle A \rangle, C]\!]_{\text{qviol}} = n_1 + n_2$.

Inductive step. Let $\pi = \langle A \rangle \circ \pi'$ with $\mathsf{Prog}(\langle A \rangle, C) \neq \varepsilon_C$. By the recursive definitions,

$$[\![\pi, C]\!]_{\text{qviol}} = [\![\langle A \rangle, C]\!]_{\text{qviol}} + \left[\!\!\left[\pi', \mathsf{Prog}(\langle A \rangle, C)\right]\!\!\right]_{\text{qviol}},$$

and

$$[\![\pi, C]\!]_{\text{qblame}} = [\![\langle A \rangle, C]\!]_{\text{qblame}} + \left[\!\!\left[\pi', \mathsf{Prog}(\langle A \rangle, C)\right]\!\!\right]_{\text{qblame}}.$$

Write

$$[\![\langle A \rangle, C]\!]_{\text{qblame}} = (a_1, a_2) \quad \text{and} \quad \left[\!\!\left[\pi', \mathsf{Prog}(\langle A \rangle, C)\right]\!\!\right]_{\text{qblame}} = (b_1, b_2).$$

By the induction hypothesis,

$$\left[\!\!\left[\pi', \mathsf{Prog}(\langle A \rangle, C)\right]\!\!\right]_{\text{qviol}} = b_1 + b_2,$$

and by the base case,

$$[\![\langle A \rangle, C]\!]_{\text{qviol}} = a_1 + a_2.$$

Therefore,

$$[\![\pi, C]\!]_{\text{qviol}} = (a_1 + a_2) + (b_1 + b_2) = (a_1 + b_1) + (a_2 + b_2) = n_1 + n_2,$$

as required. $\square$

Lemma 19 (Blame Score vs. Forward Blame Semantics)**.** *Let $\pi \in \Gamma^*$ be a finite trace and $C \in \mathsf{TACNL}$ be a contract. If the forward blame monitor assigns blame to a set of agents $S \subseteq \{1,2\}$ on π, then the corresponding component(s) of the persistent quantitative blame vector are non-zero. Formally,*

$$\textit{if } [\![\pi, C]\!]_{11} = \perp^t_S \textit{ or } [\![\pi, C]\!]_{11} = \perp^p_S$$

then there exist $x, y \in \mathbb{N}$ such that

$$[\![\pi, C]\!]_{qblame} = (x, y) \quad \textit{and} \quad \begin{cases} x \neq 0 & \textit{if } 1 \in S, \\ y \neq 0 & \textit{if } 2 \in S. \end{cases}$$

7.6 Quantitative Blame Monitor Construction

The forward-looking blame monitor $\mathsf{BMC}(C)$ identifies the *first* decisive blame frontier and then remains in a post-violation sink. To account for *all* blame occurrences over the whole lifespan of an interaction, we introduce a quantitative blame monitor whose output is a pair of natural numbers counting blame assigned to each agent.

Monitor interface. We fix two agents $\{1,2\}$ and write $\mathbb{N}^2$ for blame vectors. For $\vec{n} = (n_1, n_2)$ and $\vec{m} = (m_1, m_2)$ we write

$$\vec{n} + \vec{m} := (n_1 + m_1,\ n_2 + m_2).$$

Definition 48 (Quantitative blame monitor)**.** *Let C be a contract in* TACNL *. The* quantitative blame monitor *for C is the Mealy machine*

$$\mathsf{QBM}(C) := (Q, q_0, \Gamma, \delta, \lambda_{\mathbb{N}^2}),$$

where:

1. $Q := \{\mathsf{Prog}(\pi, C) \mid \pi \in \Gamma^*\}$ *is the set of reachable residual contracts.*
2. $q_0 := \mathsf{Prog}(\varepsilon, C) = C$.
3. $\delta(q, A) := \mathsf{Prog}(\langle A \rangle, q)$.
4. $\lambda_{\mathbb{N}^2}(q, A) := [\![\langle A \rangle, q]\!]_{qblame}$*, that is, the instantaneous blame vector contributed by the current letter A when the active residual is q.*

Definition 49 (Quantitative blame execution score)**.** *Let* $\mathsf{QBM}(C) = (Q, q_0, \Gamma, \delta, \lambda_{\mathbb{N}^2})$ *and let* $\pi = \langle A_0, \ldots, A_{n-1} \rangle \in \Gamma^*$*. The* quantitative blame score *of* π *on* $\mathsf{QBM}(C)$ *is*

$$\mathsf{Score}(\mathsf{QBM}(C), \pi) := \sum_{i=0}^{n-1} \lambda_{\mathbb{N}^2}(q_i, A_i),$$

where $q_{i+1} := \delta(q_i, A_i)$ *and the sum is componentwise.*

Conformance to the quantitative blame semantics. The next theorem states that the machine-level accumulation coincides with the denotational quantitative blame semantics.

Theorem 7.3 (Conformance of QBM to quantitative blame semantics)**.** *For every* TACNL *contract C and every finite trace* $\pi \in \Gamma^*$*,*

$$\mathsf{Score}(\mathsf{QBM}(C), \pi) \;=\; [\![\pi, C]\!]_{qblame}.$$

Proof. We proceed by induction on $n := |\pi|$.

Base case $n = 0$. If $\pi = \varepsilon$, then by Definition 49 the sum is the neutral element $(0,0)$. By the definition of $[\![\cdot,\cdot]\!]_{\mathrm{qblame}}$ on the empty trace, $[\![\varepsilon, C]\!]_{\mathrm{qblame}} = (0,0)$. Hence, the equality holds.

Inductive step. Let $\pi = \langle A \rangle \circ \pi'$ with $|\pi'| = n-1$. The first output produced by $\mathsf{QBM}(C)$ is $\lambda_{\mathbb{N}^2}(q_0, A) = [\![\langle A \rangle, C]\!]_{\text{qblame}}$ by Definition 48. The next state is $q_1 = \delta(q_0, A) = \mathsf{Prog}(\langle A \rangle, C)$. Therefore, unfolding Definition 49 yields

$$\mathsf{Score}(\mathsf{QBM}(C), \pi) = [\![\langle A \rangle, C]\!]_{\text{qblame}} + \mathsf{Score}(\mathsf{QBM}(q_1), \pi').$$

By the induction hypothesis applied to the residual contract q_1, we have

$$\mathsf{Score}(\mathsf{QBM}(q_1), \pi') = [\![\pi', q_1]\!]_{\text{qblame}}.$$

Hence

$$\mathsf{Score}(\mathsf{QBM}(C), \pi) = [\![\langle A \rangle, C]\!]_{\text{qblame}} + [\![\pi', \mathsf{Prog}(\langle A \rangle, C)]\!]_{\text{qblame}}.$$

Finally, this is exactly the recursive clause of the quantitative blame semantics: it evaluates the head letter against the current residual, then propagates the resulting residual to the tail. Thus $\mathsf{Score}(\mathsf{QBM}(C), \pi) = [\![\pi, C]\!]_{\text{qblame}}$. $\square$

The quantitative blame setting combines the persistence of the quantitative monitors with the responsibility granularity of the blame verdicts. Instead of outputting a Boolean or multi-valued status, the monitor outputs a *blame increment vector* in $\mathbb{N}^2$ at each step, where the i-th component counts how many violations at that step are blamed on agent i. The overall quantitative blame score is obtained by summing these vectors along the run.

Example 39 (Quantitative Blame Monitor for Double Blame)**.** *We construct the quantitative blame monitor for the contract*

$$C := \mathbf{P}_1(\mathsf{OCC}) \wedge \mathbf{O}_1(\mathsf{PAY\_R}),$$

where $\mathbf{P}_1(\mathsf{OCC})$ models the tenant's power to occupy. As in Example **??***, the first step fully determines whether a violation occurs. However, in the quantitative blame view, the transition is labeled with a vector in $\mathbb{N}^2$ that counts* how many *violations occurred at that step and* who *is responsible.*

We use the following event classes (as shorthand predicates over a letter $A \subseteq \Gamma$):

- $\mathsf{OCC}^{\checkmark}$*: occupation succeeds (tenant attempts and landlord cooperates),*
- $\mathsf{OCC}^{\times}$*: occupation is blocked (tenant attempts, landlord withholds cooperation),*
- $\mathsf{PAY\_R}^{\checkmark}$*: rent payment succeeds (joint execution present),*
- $\mathsf{PAY\_R}^{\mathit{fail}}$*: tenant does not attempt to pay ($\mathsf{PAY\_R}^{(1)} \notin A$),*
- $\mathsf{PAY\_R}^{\mathit{blk}}$*: tenant attempts to pay but the landlord blocks cooperation ($\mathsf{PAY\_R}^{(1)} \in A \wedge \mathsf{PAY\_R}^{(2)} \notin A$).*

The crucial point is that the conjunction is additive *at a time point. Hence, when both conjuncts are violated in the same step, we add their blame contributions component-wise. In particular, the joint failure $\mathsf{OCC}^{\times} \wedge \mathsf{PAY\_R}^{\mathit{fail}}$ yields $(1,1)$, while the joint failure $\mathsf{OCC}^{\times} \wedge \mathsf{PAY\_R}^{\mathit{blk}}$ yields $(0,2)$ since both violations are blamed on agent 2.*

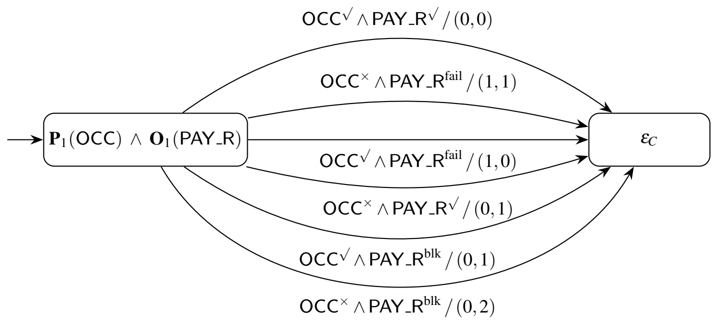

$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\checkmark} / (0,0)$
$\mathsf{OCC}^{\times} \wedge \mathsf{PAY\_R}^{\text{fail}} / (1,1)$
$\mathbf{P}_1(\mathsf{OCC}) \wedge \mathbf{O}_1(\mathsf{PAY\_R})$
ε_C
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\text{fail}} / (1,0)$
$\mathsf{OCC}^{\times} \wedge \mathsf{PAY\_R}^{\checkmark} / (0,1)$
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\text{blk}} / (0,1)$
$\mathsf{OCC}^{\times} \wedge \mathsf{PAY\_R}^{\text{blk}} / (0,2)$

Figure 26: Quantitative blame monitor fragment for $\mathbf{P}_1(\mathsf{OCC}) \wedge \mathbf{O}_1(\mathsf{PAY\_R})$. The label $/(n_1, n_2)$ is the instantaneous blame increment vector. Note the two joint-failure cases: $(1,1)$ when the violations are blamed on different agents, and $(0,2)$ when both violations are blamed on agent 2 in the same step.

7.7 Conclusion and Limitations

In this section, we established a quantitative semantics that transforms the verification of open-ended contracts from a binary decision into a cumulative cost assessment. By synthesizing the contract logic into a deterministic Moore machine, we achieved a monitor that outputs granular, instantaneous violation scores. This enables a more refined dispute-resolution process in which penalties are proportional to the severity and frequency of non-compliance, rather than a simple pass/fail verdict.

However, our current scoring mechanism abstracts away various important distinctions found in real-world legal systems. A primary limitation is that the semantics currently conflate the cost of a violation with the cost of a reparation. In our model, a non-zero score simply indicates that the ideal path was not taken, without distinguishing whether the agent is paying a penalty (secondary obligation) or persistently violating the contract. Legal theory emphasizes the fundamental difference between *primary rules* of obligation and *secondary rules* of recognition and adjudication as show in [11]. To be consistent with this legal reality, our logic requires a refinement where the reparation operator $C \blacktriangleright C'$ treats the violation of C differently from the execution of C'. This could be achieved by introducing a distinct "reparation score" or by masking the violation score of C when C' is successfully executed, ensuring that "repaired compliance" is semantically distinct from "unrepaired violation."

Furthermore, our framework assumes that all failures are attributable to the agents. Real-world contracts regularly experience the *impossibility of performance* due to unenforceable events (e.g., force majeure), such as natural disasters or regulatory changes that render performance illegal [1]. Currently, our model penalizes an agent for failing to act, even if the environment prevents the action. To address this, next versions of the logic must extend the trace model to include a third component: an *environment trace*. This would allow the semantics to distinguish between unwillingness to perform (fault) and inability to perform (impossibility). Such a distinction opens the door to new verdict types, such as "shared loss" or "frustration of purpose," where the burden of reparation is distributed between agents rather than assigned to a single defaulter.

Finally, the current reparation operator is agnostic regarding the source of the failure. In a complex contract $C_1 \blacktriangleright C_2$, the secondary contract C_2 is triggered regardless of which specific clause in C_1 was violated or which agent was responsible. This does not reflect legal practice, where the remedy always depends on the specific breach and the party at fault. A necessary extension of this work requires a *parametrized reparation operator* that assigns different secondary obligations depending on the specific cause of the failure. For instance, if a joint project fails because Agent A did not pay, the reparation should differ from that if Agent B did not work. Distinguishing these cases requires a richer syntax that propagates blame information into the reparation phase.

References

[1] Taylor v. Caldwell. 3 B. & S. 826, 122 Eng. Rep. 309 (Q.B.), 1863. Foundational English law case establishing the doctrine of frustration and impossibility of performance.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[4] Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018.

[5] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM (JACM)*, 11(4):481–494, 1964.

[6] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.

[7] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[8] Yliès Falcone, Jean-Claude Fernandez, and Laurent Mounier. Runtime verification of safety-progress properties. In *Runtime Verification (RV 2009)*, volume 5779 of *Lecture Notes in Computer Science*, pages 40–59. Springer, 2009.

[9] Guido Governatori and Zoran Milosevic. A formal analysis of a business contract language. *Information Systems Frontiers*, 8(3):273–302, 2006.

[10] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.

[11] Herbert L. A. Hart. *The Concept of Law*. Clarendon Press, Oxford, 1961. Seminal text introducing the union of primary and secondary rules.

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

[13] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, MA, 2 edition, 2001.

[14] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[15] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[16] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[17] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, Boston, MA, 3 edition, 2012.

[18] Ken Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.