remastered chap3

kharrazkaram

September 2025

Contents

1 Motivations

In this section, we are interested in normative specification in multiagent scenarios. Our vision in this work is that normative specification, and more precisely, what we will refer to as contracts in this section, specify *collaborative* instead of adversarial behaviors: the agents are not playing zero-sum games, but instead must coordinate to achieve successful interactions. We limit our reasoning to the case of two-agent settings, which already suffices to capture many core difficulties of collaboration.

Interactions and collaboration. Norms typically prescribe when an agent must, may, or should abstain from performing, yet their successful execution may depend on the cooperation with another agent. In the real world, examples abound: in a house rental contract, a landlord and tenant maintaining a lease, or two drivers yielding at an intersection. To capture such interactions, we need an abstraction that makes explicit when success requires joint effort or the non-interference of one of the parties to fail the efforts of the other.

Infinite normative interactions. Normative specifications rarely describe one-off events; they often govern interactions that potentially persist indefinitely. A paradigmatic example is a rental agreement: the tenant must pay rent each month, and the landlord must provide housing services in return. Such contracts generate a potentially unbounded interaction of obligations and permissions, motivating a formalism that can reason about potentially unbounded behavior.

The Challenge of Blame. Most importantly in this chapter is that when norms are violated, the question of *blame* arises: who is responsible for the failure of collaboration? This is a notoriously subtle notion, intersecting with debates in law, philosophy, and multiagent reasoning. Is an agent to blame for not attempting its part of a joint action? Or does responsibility shift when the other agent obstructs fulfillment? Our aim in this work is to study how blame can be assigned systematically within a formal framework, without appealing to moral or psychological notions of responsibility.

Research questions. These considerations raise several questions that guide this work:

1. How can we design a logic that treats collaboration between agents as a first-class object, rather than as an adversarial competition?
2. What abstractions are best suited to capture cooperative and non-interfering behavior in normative specifications?
3. How can we model normative specifications that open-ended streams of obligations, such as ongoing contracts?
4. How should we define and reason about blame in a way that is both conforming with the legal domain and computationally tractable?

2 Motivating Example

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% of the monthly rent shall be assessed.*
- *(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.*

Features Suggested by the Example

The above clauses illustrate several requirements that a specification language for contracts should capture:

- **Mutual collaboration** (Clauses C1–C2): compliance requires both tenant and landlord to act together, e.g. one must pay while the other accepts, one must request occupancy while the other grants it.

- **Reparation mechanisms** (Clause C3): violations do not simply collapse the agreement but trigger compensatory obligations such as late fees.
- **Triggered obligations** (Clause C4): some duties arise only when one party takes an enabling step, e.g. the landlord's duty to repair is activated when the tenant submits a formal request.
- **Termination powers** (Clause C5): contracts may be unilaterally ended by one party, which reconfigures the obligations of both (continued payment, continued occupancy, eventual vacating).
- **Repetition pattern**: all obligations are periodical, repeating monthly, which requires temporal representation.

Interpretation and Research Gap

The clauses also exemplify deeper normative concepts:

- **Obligations and reparations:** some clauses express primary duties (e.g. rent payment) with secondary consequences if breached (late fees). This reflects contrary-to-duty structures.
- **Hohfeldian powers:** termination illustrates how one agent can unilaterally reshape the normative configuration. By exercising the power to give notice, the tenant changes the duties of both parties, forcing continued payment and occupancy permission until the end date, followed by mandatory vacating.
- **Triggered duties:** the repair clause shows how discretionary acts (requesting repairs) activate new obligations for the other party. Unlike reparations, these are not responses to violations but to norm-triggering events.

Taken together, the contract reflects an *open-ended setting*: agents are not locked into a fixed pattern of compliance but can change the normative landscape through violations, triggers, or unilateral powers. This stands in contrast with most logical frameworks for agency, which assume either (i) a single agent against an environment, or (ii) adversarial, zero-sum games. Realistic contracts are instead collaborative: compliance is only possible if both agents act together, and breaches or triggers can shift obligations without terminating the agreement outright.

Summary

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

1. represent the strategies of multiple agents acting in tandem rather than in opposition,
2. express obligations, powers, reparations, and triggers in a uniform framework,
3. capture open-ended dynamics where obligations evolve over time, and

4. support 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.

3 Preliminaries and State of the Art

In the remainder of the chapter we instantiate this abstract notion of trace with logical-time, metric-timed, and synchronous-time traces over suitable structured alphabets.

3.1 Traces and Their Synchronization Mechanisms

Traces (also called *executions*, *runs*, or *behaviors*) form the semantic backbone for describing how systems evolve over time. They serve as *abstract representations* of a system's evolution and may take the form of *sequences of discrete events* or *sets of continuously varying signals*. Such traces are used to determine whether the modeled or observed behavior of a system *satisfies* a given property, and they also function as *witnesses* demonstrating the feasibility of compliant or desirable behaviors against the target properties.

This section presents a state-of-the-art overview of the main trace notions used in modeling and verification, with the specific aim of clarifying the two trace models employed in this dissertation. We introduce precise definitions, explain the contexts in which each trace type appears, and highlight their differences through focused examples. Particular emphasis is placed on the two *discrete trace models over atomic actions* that underpin the technical developments of this work.

Traces and related notions appear across many areas of artificial intelligence and formal verification, often under different names that emphasise specific semantic viewpoints. This subsection provides a brief taxonomy of these notions as they are used in the literature, focusing on how different terminologies converge on a common semantic foundation. We intentionally avoid discussing the internal structure of events, since this will be addressed in the next subsection.

Runs. In automata theory and the semantics of reactive systems, a *run* is an execution of an abstract machine such as an automaton or transition system. A run determines a behaviour by following the transitions allowed by the underlying model. Runs form the core semantic notion in classical automata-theoretic model checking [29, 8]. Every run induces a (linear-time) behaviour by projecting the visited configurations to their observable components, making runs the operational counterpart of trace-based views of system behaviour.

Executions. In operational semantics and concurrency theory, an *execution* refers to the (often maximal) evolution of a system, typically formalised as a path in a labelled transition system. Executions serve as the semantic basis of process calculi, labelled

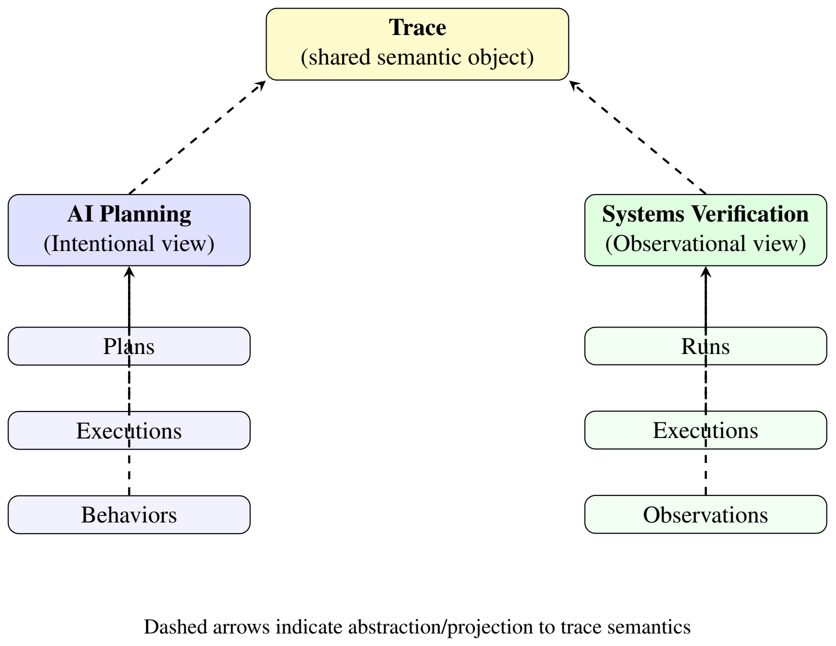


Figure 1: Conceptual alignment of trace notions across AI planning and systems verification. Arrows indicate abstraction relationships between domain-specific concepts and the shared notion of traces.

transition systems, and state-based verification frameworks [24, 17]. Although richer than purely observational accounts, executions can be abstracted to trace-like objects when only the externally relevant behaviour is considered, for instance by hiding internal actions or projecting to observable variables.

Behaviours. The term *behaviour* is used to denote any admissible evolution permitted by a system model. Depending on context, behaviours may correspond to runs, executions, or directly to sequences of observable states or events. Behavioural models appear prominently in the semantics of reactive and concurrent systems, where sets of behaviours are used to characterise implementations and specifications, and to define refinement and conformance relations [16, 22, 1]. The behavioural viewpoint abstracts away from internal structure and focuses on the admissible patterns of interaction with the environment.

Plans. In AI planning, a *plan* is a finite, intended evolution of an agent or system. Plans are typically treated as finite sequences of actions that describe how a goal should be achieved, and are central in classical planning languages such as STRIPS [11] and PDDL [12]. At execution time, plans generate concrete behaviours of the underlying transition system. These behaviours are finite, in contrast to the infinite computations

Table 1: Correspondence of trace-related terminology across AI planning and systems verification

Domain	**Term**	**Typical Length**	**Level of Abstraction**
Planning	Plan	Finite	Intentional (goal-directed)
	Plan Execution	Finite	Operational
	Behaviour	Finite/Infinite	Observable
Verification	Run	Infinite	Operational (automaton)
	Execution	Finite/Infinite	Operational (transition system)
	Observation	Finite prefix	Partial information
Common	Trace	Finite/Infinite	Observable (projection)

usually considered in reactive system verification; nevertheless, they can be related to linear-time formalisms, for example via finite-trace temporal logics such as LTL_f [13].

Observations. An *observation* is a fragment of behaviour available to an external monitor or agent. Observations are often incomplete: they may arise from projections onto a subsystem, from hiding of internal actions, or from partial information about states. This notion is essential in runtime verification [21], epistemic reasoning about knowledge and information flow [9], and distributed monitoring [5]. Observations highlight the distinction between the full behaviours a system may exhibit and the partial behavioural evidence actually accessible to an analyst or monitoring component.

Table 1 summarises the correspondences between these notions across the two domains.

We now move to an abstract definition of traces that captures their essential structure independently of these domain-specific terminologies.

3.1.1 Abstract Trace Theory

Informal overview. Traces are mathematical objects that record how a system evolves over time. Two fundamental forms appear across logic, semantics, and verification. *Discrete traces* describe executions as sequences of events indexed by natural numbers. *Continuous traces* describe behaviors as signals evolving over real time. Both serve as abstractions of system behavior, yet they rely on very different underlying notions of time.

Events and event domains. A single element that occurs during an execution is called an *event*. Events do not carry a prescribed structure. They may represent system actions, state changes, observations, valuations, or abstract labels. An event set can be instantiated by any concrete *event domain* $\mathbb{E}$, which allows the trace definitions below to remain general and domain neutral.

Definition 1 (Discrete traces)**.** *Let* $\mathbb{E}$ *be an event domain. A* finite trace *over* $\mathbb{E}$ *is either the empty trace* $\langle - \rangle$ *or a sequence*

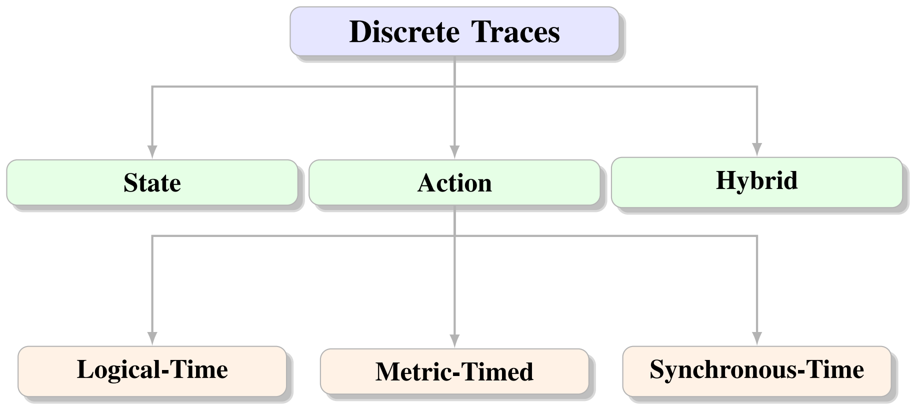


Figure 2: Classification of discrete traces following there event types and for the action, the sub-classes following their timing setup.

$\langle e_0, e_1, \ldots, e_{n-1} \rangle$ *with* $n \geq 1,\ e_i \in \mathbb{E}$. *We write the* domain of finite discrete traces*, written* $\mathbb{E}^*$ *as:*

$$\mathbb{E}^* := \{\langle - \rangle\} \cup \{\, \langle e_0, \ldots, e_{n-1} \rangle \mid n \geq 1,\ e_i \in \mathbb{E}\,\}.$$

A infinite trace *over* $\mathbb{E}$ *is a function* $\tau_{inf} : \mathbb{N} \to \mathbb{E}$*, equivalently written as* $\langle e_0, e_1, e_2, \ldots \rangle$*, and* the set of all infinite traces *is* $\mathbb{E}^\omega$*. The full discrete trace space is*

$$\mathbb{E}^\infty := \mathbb{E}^* \cup \mathbb{E}^\omega.$$

Trace length. The size of a trace τ from E^∞ is written as $|\tau|$ and is defined as the number of events in the trace. That is: For any finite non-empty trace $\tau = \langle e_0, e_1, \ldots, e_{n-1} \rangle$, we have $|\tau| := n$.

For the empty trace $\langle - \rangle$, the size is defined as: $|\langle - \rangle| = 0$. For any infinite trace τ_{inf} from $\mathbb{E}^\omega$, we have $|\tau_{\text{inf}}| = \infty$.

Position lookup. For any trace $\pi \in \mathbb{E}^\infty$ we define a partial *position lookup function*

$$\mathsf{pos}(\pi, i) \;:=\; \begin{cases} e_i, & \text{if } \pi = \langle e_0, \ldots, e_{n-1} \rangle \text{ with } 0 \leq i < n, \\ e_i, & \text{if} \pi = \langle e_0, \ldots, e_{n-1}, \ldots \rangle, \\ \mathsf{undefined}, & \text{otherwise.} \end{cases}$$

It returns the event at index i whenever this index exists. Finite traces provide events only up to their last position. Infinite traces provide an event at every natural index.

Event precedence axiom. Discrete traces impose a strict temporal ordering on their events. For any trace $\pi \in \mathbb{E}^\infty$, if $\mathsf{pos}(\pi, i) = e_i$ and $\mathsf{pos}(\pi, j) = e_j$ with $i < j$, then e_i *precedes* e_j in the execution. This precedence relation reflects the intrinsic time structure of discrete traces: earlier indices represent earlier steps, and no two different events share the same position.

Prefix notations. For any trace $\tau \in \mathbb{E}^\infty$ and any index $k \in \mathbb{N}$ we define *the finite prefix up to k*, written $\tau[0,k]$ by

$$\tau[0,k] := \begin{cases} \langle e_0, e_1, \ldots, e_k \rangle, & \text{if } \tau = \langle e_0, \ldots, e_{n-1} \rangle \text{ with } k < n, \\ \langle e_0, e_1, \ldots, e_k \rangle, & \text{if } \tau = \langle e_0, e_1, \ldots \rangle \text{ (infinite trace)}, \\ \mathsf{undefined}, & \text{otherwise.} \end{cases}$$

Thus, $\tau[0,k]$ yields the finite list of all events in τ from position 0 to position k, whenever these positions exist. Finite traces admit prefixes only up to their last valid index, while infinite traces admit prefixes for every natural number.

For two traces τ and τ' over $\mathbb{E}^\infty$ we say that τ *is a prefix* τ', written $\tau \preceq \tau'$

$$\tau \preceq \tau' \quad \text{iff} \quad \tau = \tau'[0,k] \text{ for some } k \in \mathbb{N}.$$

Suffix notation. For any trace $\pi \in \mathbb{E}^\infty$ and any index $k \in \mathbb{N}$ we define the suffix π^k by

$$\pi^k := \begin{cases} \langle e_k, e_{k+1}, \ldots, e_{n-1} \rangle, & \text{if } \pi = \langle e_0, \ldots, e_{n-1} \rangle \text{ and } k < n, \\ \langle e_k, e_{k+1}, \ldots \rangle, & \text{if } \pi = \langle e_0, e_1, \ldots \rangle \text{ (infinite trace)}, \\ \mathsf{undefined}, & \text{otherwise.} \end{cases}$$

Thus, π^k returns the portion of the trace that begins at position k when this position exists.

Informal view on continuous behavior. Continuous behavior does not rely on sequences of events. It models systems that evolve through uninterrupted time, such as physical processes, hybrid dynamics, or real-time controllers. The mathematical primitive is not a sequence but a function that assigns a value to each real-valued non-zero instant, i.e time-points from $\mathbb{R}_{\geq 0}$. Values may represent system states, sensor readings, or continuous variables.

Definition 2 (Continuous traces)**.** *Let V be a value domain. A* continuous trace $\widetilde{\tau}$ *is a function*

$$\widetilde{\tau} : \mathbb{R}_{\geq 0} \to V,$$

mapping each real-valued time instant to a value in V.

Lifting the discrete trace operators to continuous traces is non-trivial, because the underlying time domain changes from the countable order $\mathbb{N}$ to the dense, uncountable order $\mathbb{R}_{\geq 0}$. Simple combinators such as prefix, suffix, and concatenation rely on a notion of "position" given by natural-number indices and on traces being built from finite or ω-sequences of events. For continuous traces $\pi : \mathbb{R}_{\geq 0} \to V$, there is no last position and no canonical next point in time, so discrete prefix/suffix operators have to be replaced by restriction to intervals, time-shifts, and composition on real domains, typically formulated in terms of signal or trajectory operators from hybrid systems and signal temporal logic [19, 23, 7]. This change of time model also forces temporal operators and monitoring algorithms to account for dense-time semantics and uncountably many potential discontinuities, making direct reuse of the discrete machinery impossible in general.

Formal verification of software systems focuses primarily on abstraction for discrete traces because algorithms, programs, and protocol interactions proceed through countable computational steps. This dissertation adopts this perspective and works exclusively with discrete traces.

We now introduce the different concrete event domains used in the remainder of the chapter.

4 Discrete Trace Taxonomy

Discrete traces differ according to the choice of event domain $\mathbb{E}$.

I am confused help i need also to say that event are rich data structures i.e vector of other data structure but mainly I want to introduce action, states and state action

Action Traces Action traces have , a set of elementary actions or transition labels. Each event denotes a computational step or interaction. This model underlies labeled transition systems, process algebras such as CCS and CSP, and classical trace theory.

State Traces State traces use 2^{AP}. Each event is a full state valuation describing which propositions hold at that step. This is the semantic model for Kripke structures and temporal logics such as LTL and CTL.

Hybrid State–Action Traces Hybrid traces combine states and actions, either by pairing events in $S \times A$ or by alternating states and actions $s_0, a_0, s_1, a_1, \ldots$. These traces arise in operational semantics, debugging, and model checker counterexample explanations.

4.1 Comparison of Trace Types

Trace Type	Event Meaning	Semantic Focus	Frameworks
Action	Transition steps	Control flow	LTS, CCS, CSP
State	Propositional valuations	Temporal truth	Kripke structures, LTL
State–action	Cause and resulting state	Operational causality	Operational semantics, SPIN

In this dissertation we focus on *action traces*, where events are drawn from a finite alphabet of actions. We work with per-agent action alphabets Σ_i and write $\Sigma := \bigcup_i \Sigma_i$ for the global alphabet. The literature has converged on three canonical action trace models, each embodying a different view of time: *metric-timed traces*, *logical-time traces*, and *synchronous-time traces*. Metric-timed traces are standard in real time verification and timed automata [2]; logical-time traces supports interleaving models for process calculi and concurrency [25, 17, 6]; and synchronous-time traces are used in synchronous languages and round-based control [15]. For each model we describe:

1. the event domain and the motivation in practice,
2. the notion of timed distance between events that is different for each type of action trace, and
3. a synchronization operator that combines two traces of the same type.

To talk about temporal timed distance between event in a uniform way we define different variants of a partial *timed distance* function

$$\mathsf{tdist} : \mathbb{E} \times \mathbb{E} \rightharpoonup \mathbb{Z},$$

whose concrete definition depends on the type of the trace. We now instantiate this scheme for the three trace models of interest.

4.1.1 Metric-Timed Action Traces

Motivation and usage. Metric-timed traces record which action occurs and at which discrete time point. They are the staple model for real time systems, timed automata, and timed process theories, where absolute time and quantitative delays are semantically relevant [2]. Typical applications include scheduling constraints, deadlines, and bounded response obligations.

Event domain. Given an action alphabet Σ, the metric-timed event domain is the set of action–timestamp pairs

$$\mathbb{E}_{\mathrm{mt}} := \Sigma \times \mathbb{N}.$$

Definition 3 (Action Lookup Function by timestamp)**.** *The* action lookup function ρ : $\mathbb{E}^*_{mt} \times \mathbb{N} \to \Sigma \cup \{\mathsf{undefined}\}$ *returns the action* a *performed at time t in trace τ, if any:*

$$\rho\big(\langle (a_1,t_1)\dots(a_n,t_n)\rangle, t\big) := \begin{cases} a_i & \textit{if } t_i = t \textit{ for some } i, \\ \textit{undefined} & \textit{otherwise.} \end{cases}$$

and for the empty trace, $\rho(\langle - \rangle, t) :=$ *undefined*

Definition 4 (Timed distance for two metric timed events from the same trace)**.**

$$\textit{Let } \tau^{\mathrm{mt}} = \langle (a_0,t_0),(a_1,t_1),\dots,(a_{n-1},t_{n-1})\rangle \in (\Sigma \times \mathbb{N})^*$$

we define the distance between any two events from the metric timed trace as tdist_{mt} : $\mathbb{E}_{mt} \times \mathbb{E}_{mt} \to \mathbb{Z}$ *:*

$$\mathsf{tdist}_{mt}\big((a_i,t_i),(a_j,t_j)\big) := t_j - t_i$$

whenever $t_j \geq t_i$. For metric-timed traces this yields the intuitive timed distance between any two events on the same trace. Metric-timed synchronization captures simultaneous behavior of two agents/systems that share the same time scale and act at concrete timestamps.

We move now to define the classical synchronization mechanism which relies on *the global clock assumption*

Definition 5 (Synchronization of two timed traces)**.**

$$\textit{Let } \tau_1^{\mathrm{mt}} \in (\Sigma_1 \times \mathbb{N})^* \quad \textit{and} \quad \tau_2^{\mathrm{mt}} \in (\Sigma_2 \times \mathbb{N})^*$$

be two metric-timed traces defined by the same global clock. Their synchronized trace

$$\tau_1^{\mathrm{mt}} \,||_{\mathrm{mt}}\, \tau_2^{\mathrm{mt}}$$

is the (finite) trace over $(\Sigma_1 \times \Sigma_2) \times \mathbb{N}$ *that pairs up events with the same timestamp:*

$$\tau_1^{\mathrm{mt}} \,||_{\mathrm{mt}}\, \tau_2^{\mathrm{mt}} \;:=\; \langle ((a_{k_0}^{(1)}, a_{\ell_0}^{(2)}), t_0), \ldots, ((a_{k_m}^{(1)}, a_{\ell_m}^{(2)}), t_m) \rangle$$

where:

- *each* t_j *belongs to* $\mathsf{time}(\tau_1^{\mathrm{mt}}) \cap \mathsf{time}(\tau_2^{\mathrm{mt}})$*,*
- $(a_{k_j}^{(1)}, t_j)$ *occurs in* τ_1^{mt} *and* $(a_{\ell_j}^{(2)}, t_j)$ *occurs in* τ_2^{mt}*, and*
- *the timestamps* $t_0 < \cdots < t_m$ *enumerate* $\mathsf{time}(\tau_1^{\mathrm{mt}}) \cap \mathsf{time}(\tau_2^{\mathrm{mt}})$ *in increasing order.*

Events that do not share a timestamp remain local to the respective agent and can be handled later through projection or product constructions in the usual way for timed automata.

Example 2 (Factory robot and quality controller)**.** *Let*

$$\Sigma_1 = \{\mathit{load},\ \mathit{weld},\ \mathit{paint}\}, \qquad \Sigma_2 = \{\mathit{inspect},\ \mathit{approve},\ \mathit{reject}\}.$$

Consider the two metric timed traces

$$\begin{aligned} \tau_1^{\mathrm{mt}} &= \langle (\mathit{load}, 1),\ (\mathit{weld}, 4),\ (\mathit{paint}, 7) \rangle, \\ \tau_2^{\mathrm{mt}} &= \langle (\mathit{inspect}, 4),\ (\mathit{approve}, 5),\ (\mathit{inspect}, 7) \rangle. \end{aligned}$$

Event lookup using ρ.

$$\rho(\tau_1^{\mathrm{mt}}, 4) = \mathit{weld}, \qquad \rho(\tau_2^{\mathrm{mt}}, 5) = \mathit{approve}, \qquad \rho(\tau_1^{\mathrm{mt}}, 2) \textit{ undefined.}$$

Timed distances.

$$\mathsf{tdist}_{mt}\big((\mathit{load}, 1), (\mathit{weld}, 4)\big) = 4 - 1 = 3, \qquad \mathsf{tdist}_{mt}\big((\mathit{inspect}, 4), (\mathit{inspect}, 7)\big) = 7 - 4 = 3.$$

4.1.2 Logical-Time Action Traces

Motivation and usage. Logical-time traces abstract away absolute time and retain only the order in which actions occur. They are the underlying objects in interleaving models for process calculi such as CCS and CSP [25, 17] and in Mazurkiewicz trace theory for true concurrency [6]. This view is appropriate when only causal or ordering information matters and concrete delays are irrelevant.

Event domain. Given an action alphabet Σ, the logical-time event domain is simply

$$\mathbb{E}_{\text{lt}} := \Sigma.$$

Formal definition. A *finite logical-time trace* over Σ is a word

$$\tau^{\text{lt}} = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in \Sigma^*$$

with $a_k \in \Sigma$. We write $|\tau^{\text{lt}}| := n$ and $\mathsf{pos}(\tau^{\text{lt}}) := \{0, \ldots, n-1\}$ for the position set.

Timed distance. Since logical time discards timestamps, there is no meaningful metric distance between events. We reflect this by treating the timed distance as undefined on logical-time events:

$$\mathsf{tdist}(e_i, e_j) \text{ is undefined for all } e_i, e_j \in \mathbb{E}_{\text{lt}}.$$

Only the relative order of events is available.

Synchronization. Synchronization on logical-time traces is given by the standard asynchronous interleaving or *shuffle* operator that combines two local traces while preserving the local order of each. Let $\Sigma = \Sigma_1 \uplus \Sigma_2$ be the disjoint union of per-agent alphabets and let

$$u \in \Sigma_1^*, \qquad v \in \Sigma_2^*.$$

The asynchronous synchronization $u \,||_{\text{lt}}\, v$ is the set of all words in Σ^* obtained by interleaving u and v without reordering the letters of either argument. Formally, we define

$$u \,||_{\text{lt}}\, v := u \,|||\, v,$$

where the shuffle $u \,|||\, v$ is characterised inductively by

$$\begin{aligned}
\langle - \rangle \,|||\, v &= \{ v \}, \\
u \,|||\, \langle - \rangle &= \{ u \}, \\
(a \cdot u') \,|||\, (b \cdot v') &= \{ a \cdot w \mid w \in u' \,|||\, (b \cdot v') \} \cup \{ b \cdot w \mid w \in (a \cdot u') \,|||\, v' \},
\end{aligned}$$

with $a \in \Sigma_1$ and $b \in \Sigma_2$. Every $w \in u \,|||\, v$ satisfies the projection property

$$w \restriction \Sigma_1 = u, \qquad w \restriction \Sigma_2 = v.$$

This operator corresponds to the standard interleaving semantics used in process calculi [25, 17].

4.1.3 Synchronous-Time Action Traces

Motivation and usage. Synchronous-time traces assume a single global logical clock. All components react simultaneously at each round, and absence of action is explicit. This is the semantic basis of synchronous languages such as Lustre and Esterel and of many round-based controllers and distributed protocols with a globally synchronised step [15].

Event domain. Given an action alphabet Σ, we extend it with a dedicated stutter symbol "$-$" that is not in Σ and define

$$\mathbb{E}_{\mathrm{st}} := \Sigma \cup \{-\}.$$

At each round the event either records a concrete action from Σ or records that no action occurs.

Formal definition. Fix a global round clock indexed by $\mathbb{N}$. A *finite synchronous-time trace* over Σ is a word

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

with $s_t \in \Sigma \cup \{-\}$ the observation at round t. The length is $|\tau^{\mathrm{st}}| := T+1$ when the word is nonempty and the set of positions is $\mathsf{pos}(\tau^{\mathrm{st}}) := \{0, \ldots, T\}$.

Timed distance. We fix a positive constant $\Delta > 0$ for the duration of one global round. For synchronous-time events we interpret the timed distance by their round indices:

$$\mathsf{tdist}(s_{r_i}, s_{r_j}) := (r_j - r_i) \cdot \Delta$$

whenever $0 \leq r_i \leq r_j \leq T$. Thus the temporal separation between events is given by the number of intervening rounds multiplied by the fixed round duration.

Synchronization. Synchronization on synchronous-time traces is given by a lockstep or *zip* operator that combines two global views round by round. Let

$$\tau_1^{\mathrm{st}} = \langle s_0, \ldots, s_{T_1} \rangle \in (\Sigma_1 \cup \{-\})^*, \qquad \tau_2^{\mathrm{st}} = \langle r_0, \ldots, r_{T_2} \rangle \in (\Sigma_2 \cup \{-\})^*.$$

Set $T := \max(T_1, T_2)$ and extend the shorter word with stutters up to horizon T. The synchronous synchronization

$$\tau_1^{\mathrm{st}} \,||_{\mathrm{st}}\, \tau_2^{\mathrm{st}}$$

is the trace over $(\Sigma_1 \cup \{-\}) \times (\Sigma_2 \cup \{-\})$ defined by

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

Each position t of the result records the joint round of both agents. Projecting onto the first (respectively second) component recovers the padded version of τ_1^{st} (respectively τ_2^{st}). This is the standard lockstep product used in synchronous languages and synchronous composition of automata [15].

In summary, these three action trace models share the same abstract notion of events and traces but differ in the temporal information carried by each event, in the induced notion of timed distance, and in their native synchronization operators. We build on these models in the rest of the chapter when we introduce automata-based tools and multi-agent synchronization disciplines for collaborative normative specifications.

4.2 Model Based Logics

4.2.1 Regular Languages

4.3 ω-Regular Languages

4.4 Formal tools for trace verification

Trace-based verification reduces semantic questions about systems and specifications to questions about sets of words over an alphabet. Two central analysis tasks are model checking and runtime monitoring.

Model checking versus monitoring. In model checking, a (finite-state) model of the system is given, for example as a labelled transition system, and a specification is given as a logical formula or an automaton. Verification asks whether all executions of the model satisfy the specification, which can be reduced to language inclusion or emptiness of a product construction. In runtime monitoring, only a single concrete execution trace is available, generated by a running system. A monitor processes the trace incrementally and produces a verdict about whether the observed behaviour satisfies, violates, or is still inconclusive with respect to the specification.

In both cases, finite automata over traces are the core workhorse. They serve either as compiled forms of temporal or deontic specifications, or as abstract models of the behaviours that a system can generate. We recall the standard notions needed later.

Nondeterministic finite automata.

Definition 6 (Nondeterministic finite automaton (NFA))**.** *Let Σ be an alphabet. A nondeterministic finite automaton (*NFA*) over Σ is a tuple*

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

where:

- *Q is a finite set of states,*
- *$q_0 \in Q$ is the initial state,*
- *$F \subseteq Q$ is the set of accepting states,*
- *$\delta \subseteq Q \times \Sigma \times Q$ is a transition relation.*

A run *of $\mathscr{A}$ on a finite word $\pi = \langle a_0, \ldots, a_{n-1} \rangle \in \Sigma^*$ is a sequence of states*

$$r = \langle q_0, q_1, \ldots, q_n \rangle$$

such that $(q_k, a_k, q_{k+1}) \in \delta$ for every $k < n$. The run is accepting *if $q_n \in F$. The language recognised by $\mathscr{A}$ is*

$$\mathscr{L}(\mathscr{A}) \;:=\; \{\, \pi \in \Sigma^* \mid \textit{there exists an accepting run of } \mathscr{A} \textit{ on } \pi \,\}.$$

ε-NFA.

Definition 7 (ε-NFA). *An ε-NFA over alphabet Σ is an NFA*

$$\mathscr{A}_\varepsilon = (Q, \Sigma, \delta, q_0, F)$$

where the transition relation δ may additionally contain transitions of the form (q, ε, q') that consume no input symbol. A run of $\mathscr{A}_\varepsilon$ on a word $\pi \in \Sigma^*$ is allowed to take ε-transitions between consuming letters. An ε-NFA recognises the same class of regular languages as NFAs. Every ε-NFA can be transformed into an equivalent NFA (without ε-moves) by the standard ε-elimination construction.

Deterministic finite automata.

Definition 8 (Deterministic finite automaton (DFA)). *A deterministic finite automaton (DFA) over an alphabet Σ is a tuple*

$$\mathscr{D} = (Q, \Sigma, \delta, q_0, F),$$

where Q, Σ, q_0, and F are as above, and the transition function

$$\delta : Q \times \Sigma \to Q$$

is total and single-valued. Thus, for every state $q \in Q$ and letter $a \in \Sigma$ there is exactly one successor state $\delta(q, a)$. A run of $\mathscr{D}$ on a word $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ is the unique state sequence

$$r = \langle q_0, q_1, \ldots, q_n \rangle$$

with $q_{k+1} = \delta(q_k, a_k)$ for all $k < n$. The word π is accepted if $q_n \in F$, and the recognised language $\mathscr{L}(\mathscr{D})$ is defined accordingly.

Automata and trace verification. For a property φ over traces on alphabet Σ, the set of all traces that satisfy φ can often be represented as a regular language $L_\varphi \subseteq \Sigma^*$ accepted by some NFA or DFA. Model checking questions such as "does every execution of the system satisfy φ?" can then be reduced to language inclusion problems $L_{\text{sys}} \subseteq L_\varphi$, whereas monitoring questions "does the current finite trace satisfy φ?" are answered by running the corresponding automaton on the observed prefix. Later in this chapter, we specialise these constructions to deontic and collaborative specifications and refine them into Moore machines that produce rich verdicts instead of simple accept or reject outcomes.

4.5 Moore Machines

5 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 multiagent 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 [25, 17], true-concurrency and trace theory to separate independence from conflict [6], round-based lockstep in synchronous languages [15], 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 [9]. The choice of synchronization is not cosmetic. It clarifies what constitutes feasible joint behavior, which then 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 techniques of synchronization 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 $||^{A}_{\text{hs}}$ (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).

5.1 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 $\mathbb{N}$.

5.1.1 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 [25, 17, 6].

Definition 9 (Logical-time word, notation τ_i^{lt})**.** *A (finite) logical-time word of agent i over an alphabet Σ_i is a sequence*

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

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

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

Example 3 (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: $\tau_1^{\mathrm{lt}} \cdot \langle (\mathit{handover})\rangle = \langle (\mathit{pick}), (\mathit{handover}), (\mathit{handover})\rangle$*.*

5.1.2 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 10 (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* $\tau_i^{\mathrm{mt}}\restriction\{t \leq T\}$ (shorthand: $\tau_i^{\mathrm{mt}}\{t \leq T\}$). Inter-event gaps: $\Delta_k := t_{k+1}^i - t_k^i > 0$.

Example 4 (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: $\tau_1^{\mathrm{mt}}\{t \leq 2\} = \langle(\mathit{badge\_tapped}, 1)\rangle$, $\quad \tau_1^{\mathrm{mt}}\{t \leq 3\} = \tau_1^{\mathrm{mt}}$.
Gaps: $\Delta_0 = 3 - 1 = 2$.
Concatenation: $\tau_1^{\mathrm{mt}} \cdot \langle(\mathit{audit\_log}, 4)\rangle = \langle(\mathit{badge\_tapped}, 1), (\mathit{door\_open}, 3), (\mathit{audit\_log}, 4)\rangle$.

5.1.3 Synchronous-Time Words

Synchronous-time traces assume a *single global logical clock* (the *synchronous hypothesis*): each tick is an instantaneous round where components react simultaneously; absence of an action at a tick is explicit. This is the semantic backbone of synchronous languages such as Lustre and Esterel [15].

Definition 11 (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* $\tau_i^{\mathrm{st}}\restriction A \in (A \cup \{-\})^*$ keeps letters in A and all "$-$".
- *Logical (stutter-erasing) projection* $\mathsf{erase}_-(\tau_i^{\mathrm{st}}\restriction A) \in A^*$ 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 $0 \leq r \leq T$.

Example 5 (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:
$\tau_2^{\mathrm{st}} \cdot \langle(-)\rangle = \langle(-),(\mathit{door\_lock}),(-),(-)\rangle$*.*

5.1.4 Connecting the Models

Our objective in 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 6 (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* $\tau_i^{\mathrm{mt}} = \langle(a_0^i,t_0^i),\ldots,(a_{n_i-1}^i,t_{n_i-1}^i)\rangle$ *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)***

$$\forall \langle t_k'^{i} \rangle \textit{ strictly increasing} : \quad \mathsf{LT}\big(\langle (a_0^i, t_0'^{i}), \ldots, (a_{n_i-1}^i, t_{n_i-1}'^{i}) \rangle\big) \;=\; \mathsf{LT}(\tau_i^{\mathrm{mt}}).$$

Proof. Construction. Define

$$\mathsf{LT}(\tau_i^{\mathrm{mt}}) := \langle (a_0^i),\, (a_1^i),\, \ldots,\, (a_{n_i-1}^i) \rangle.$$

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

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

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

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

- ***Size / horizon relation***

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

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

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

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

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

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

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

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

$$\forall k < n_i - 1: \quad \#\big\{t \mid t_k^i < t < t_{k+1}^i,\ \mathsf{ST}(\tau_i^{\mathrm{mt}})[t] = -\big\} \ = \ t_{k+1}^i - t_k^i - 1.$$

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

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

- ***Equivalence to logical time***

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

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

$$\mathsf{ST}(\tau_i^{\mathrm{mt}})[t] \ = \ \begin{cases} a_k^i, & \text{if } t = t_k^i \text{ for the unique } k, \\ -, & \text{otherwise.} \end{cases}$$

Each property follows directly from this definition and the strict monotonicity of timestamps. $\square$

On non-invertibility. Both LT and ST are *many-to-one*. In particular, LT is *not* invertible: for any fixed $\tau_i^{\mathrm{lt}} \in \Sigma_i^*$, there are infinitely many τ_i^{mt} (different timestamp choices) and τ_i^{st} (different stutter patterns/horizons) such that $\mathsf{LT}(\tau_i^{\mathrm{mt}}) = \tau_i^{\mathrm{lt}}$ and $\mathsf{erase}_-\ (\tau_i^{\mathrm{st}}) \ = \ \tau_i^{\mathrm{lt}}$. Hence, one cannot reconstruct a unique metric or synchronous trace from logical time alone. (Conversely, given $\mathsf{ST}(\tau_i^{\mathrm{mt}})$ 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. We will use these morphisms to state synchronization operators once and then instantiate them uniformly across models.

5.2 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* [25, 17]. Synchronous languages assume a *single global round clock* (lockstep) [15]. 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 [20]; 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.

5.2.1 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 [25, 17]. In true concurrency theory, the same construction underlies *Mazurkiewicz traces*, where equivalence classes of interleaving are taken modulo an independence relation I [6].

Definition 12 (Asynchronous interleaving on τ^{lt})**.** *Let $\tau_1^{\text{lt}} \in \Sigma_1^*$ and $\tau_1^{\text{lt}} \in \Sigma_2^*$ be logical-time words. The* asynchronous interleaving operator $||| \,:\, \Sigma_1^* \times \Sigma_2^* \longrightarrow 2^{(\Sigma_1 \uplus \Sigma_2)^*}$ *is defined recursively by*

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

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 $w \in |||(u, v)$ satisfies the projection property:

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

Example 8 (Warehouse pick/scan, pure interleaving)**.** *Let $\Sigma_1 = \{\mathit{pick}\}$, $\Sigma_2 = \{\mathit{scan}\}$. Agent 1: $\tau_1^{\text{lt}} = \langle (\mathit{pick}), (\mathit{pick}) \rangle$. Agent 2: $\tau_2^{\text{lt}} = \langle (\mathit{scan}) \rangle$. Then*

$$\tau_1^{\text{lt}} \,|||\, \tau_2^{\text{lt}} = \Big\{ \langle (\mathit{scan}), (\mathit{pick}), (\mathit{pick}) \rangle,\ \langle (\mathit{pick}), (\mathit{scan}), (\mathit{pick}) \rangle,\ \langle (\mathit{pick}), (\mathit{pick}), (\mathit{scan}) \rangle \Big\}.$$

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

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 $\uplus$.

5.2.2 Synchronous (Lockstep) Operator on Synchronous Time

he *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 [15], in which all components react simultaneously at each logical tick.

Definition 13 (Lockstep zip on τ^{st})**.** *Let* $\tau_1^{\text{st}} = \langle (s_0), \ldots, (s_{T_1}) \rangle$ *and* $\tau_2^{\text{st}} = \langle (r_0), \ldots, (r_{T_2}) \rangle$ *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*

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

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

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

5.2.3 Synchronous Operator With Handshake Actions

In many systems, certain actions can only be executed *jointly* and must occur in *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) [17], enforces handshake actions as simultaneous and mutually synchronized events.

Definition 14 (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*

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

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

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

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

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

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 15 (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*

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

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

Example 9 (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 $\tau_1^{\mathrm{st}} \parallel_{\mathrm{hs}}^{A} \tau_2^{\mathrm{st}} = \langle (\mathit{pick}, -),\ (-, \mathit{scan}),\ (\mathit{handover}, \mathit{handover}) \rangle$*, and*
  $\mathsf{coll}_A(\tau_1^{\mathrm{st}} \parallel_{\mathrm{hs}}^{A} \tau_2^{\mathrm{st}}) = \langle (\mathit{pick}^{(1)}),\ (\mathit{scan}^{(2)}),\ (\mathit{handover}) \rangle$.

- 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* $\tau_1^{\mathrm{st}} \parallel_{\mathrm{hs}}^{A} \tau_2^{\mathrm{st}}$ *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:* $\tau_1^{\mathrm{st}} \parallel_{\mathrm{hs}}^{A} \tau_2^{\mathrm{st}} = \langle (-,-),\ (\mathit{pick}, \mathit{scan}),\ (-,-) \rangle$*. 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 $\parallel\!\mid$ captures all globally feasible interleavings that respect local order, without assuming any global notion of time or coordination. By contrast, the lockstep operator $\parallel_{\mathrm{lock}}$ assumes a shared global clock: it aligns traces round by round and makes stuttering explicit. Finally, $\parallel_{\mathrm{hs}}^{A}$ 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 are the state-of-the-art operators used in computer science to define the semantics of modal/temporal/strategic operators over multi-agent executions under different synchrony assumptions. In the next section, we suggest a new model more suited for reasoning about normative systems as a collaborative specification. We will reuse some aspects of the Synchronous operator with handshake actions.

5.3 First Contribution: Attempted/Successful Abstraction

5.3.1 What Do We Want to Model?

Our goal is to capture two complementary facets of *collaboration* between agents that standard system-centric models overlook, following our 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 cut_power, change_lock, or 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 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.

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

Example 10 (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 for this month, the tenant does not pay; the landlord does not perform any action for this month. 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 $I_0, I_1, I_2, I_3, \ldots$ 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 16 (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*

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

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*:*

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

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, the aggregation has a cost as it forgets intra-period order, multiplicity of actions, and exact timestamps. Therefore, selecting the appropriate period normalization is crucial.

Example 11 (Transforming timed words into periodic set traces, continued from Example. 10)**.** *By fixing the period to 30, we can decompose the 120 days to 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.

5.3.3 The Abstraction of Attempted/Declined Interactions Over Synchronized Alphabets

We introduce here the second aspect of our abstraction, which is the notion of interaction alphabets. We first merge the action alphabets of both agents into a unique one and then encode in the trace that when a is an event, it means either an agent has instantiated or accepted to collaborate on it. On the other hand, when an action is absent from the event, it means that the agent either did not do it or that he actively did another action to prevent its success. We begin by demonstrating on our example how this abstraction maintains the same meaning while reducing the number of action types. Then we demonstrate an operator to compute the successful interaction at each period of the synchronized trace. This transformation cannot be automated as long as the set of enabling and interfering actions is not explicitly stated in the normative specification. If it is not the case, the digitization engineer has to define them manually, as we are doing 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 5.2. Instead, we define a many-to-one *abstraction* from concrete per-agent actions to the collaboration alphabet Σ_C. On the tenant side, pay_rent and pay_rent_f are enablers because they instantiate the tenant's contribution toward PAY_R and PAY_F; start_occ is an enabler for OCC because it is the tenant's side of taking possession. A chargeback return_payment and return_payment_f is blocking: it nullifies the very transfer that PAY_R and *PAY_F* rely on. On the landlord side, ack_pay enables PAY_R, while grant_occ enables OCC by authorizing access. In contrast, ref_pay blocks PAY_R even if the tenant initiated payment, and change_lock, cut_power, or 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 $\mathsf{start\_occ}$ 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 12 (Transforming the periodic synchronized words over Σ_1, Σ_2 to periodic synchronized words over Σ_C)**.** *Using the traces from Example 11,*

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

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

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* $\mathsf{PAY\_R}$*, and agent 2 allows* OCC *and accepts* $\mathsf{PAY\_R}$*. In* I_1*, agent 1 pays the rent and occupies the flat and agent 2* *contributes to* $\mathsf{PAY\_R}$ *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* $\mathsf{PAY\_R}$ *and denies* OCC*, although the tenant wants to keep occupying the flat and pays.*

5.3.4 Successful Action Computation From Two Agents' Interaction

Definition 17 (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 13 (Successful collaboration)**.** *From the two transformed traces* π_1^C *and* π_2^C *from the Example.12 We illustrate* $\mathrm{Succ}(\pi_1^C,\pi_2^C)$ *in Figure. 3*

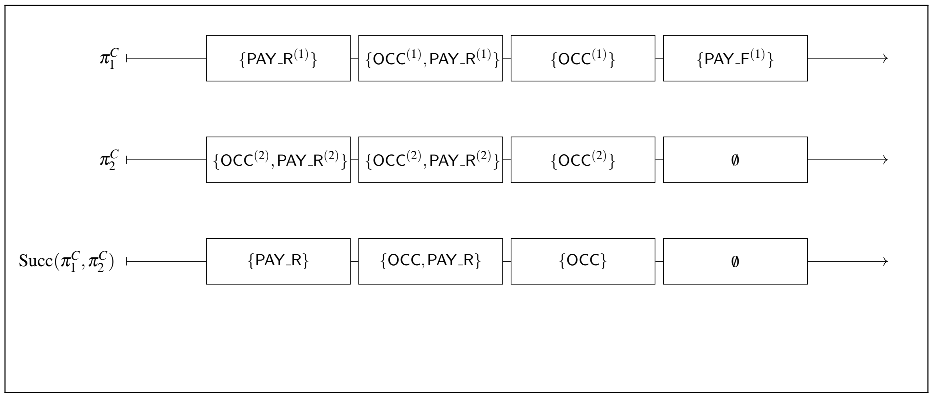

π_1^C
$\{\mathsf{PAY\_R}^{(1)}\}$
$\{\mathsf{OCC}^{(1)},\mathsf{PAY\_R}^{(1)}\}$
$\{\mathsf{OCC}^{(1)}\}$
$\{\mathsf{PAY\_F}^{(1)}\}$
π_2^C
$\{\mathsf{OCC}^{(2)},\mathsf{PAY\_R}^{(2)}\}$
$\{\mathsf{OCC}^{(2)},\mathsf{PAY\_R}^{(2)}\}$
$\{\mathsf{OCC}^{(2)}\}$
$\emptyset$
$\mathrm{Succ}(\pi_1^C,\pi_2^C)$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

Figure 3: 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* $\mathsf{PAY\_R}$*,* $\mathsf{PAY\_F}$*,* 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 meet:*

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

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

5.4 Second Contribution: Modeling Agent Interaction Strategies

Instead of reconstructing compliance *post hoc* from past events, we model how agents *intend* will behave in the future. Basically, they may want to run their intended strategies against the contract to determine whether it is compliant to prevent future violations and how other parties of the contract can potentially obstruct or abuse it in their favor.

Because collaborative success depends on both sides, each agent devises a strategy conditioned on the (observed) behavior of the other, for example, the landlord plans differently depending on whether the tenant pays; symmetrically, the tenant may stop paying if the landlord prevents continued occupancy or fails to repair the signaled damages.

5.4.1 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 18 (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 19 (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 18. 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.

5.4.2 Interactive strategy computation

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

Definition 20 (Complementary Moore machines)**.**

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

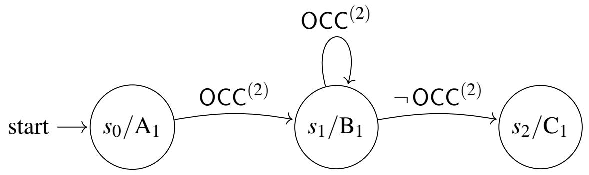
be two deterministic Moore machines, we say that M_i and *M_j* are complementary *if and only if:*

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

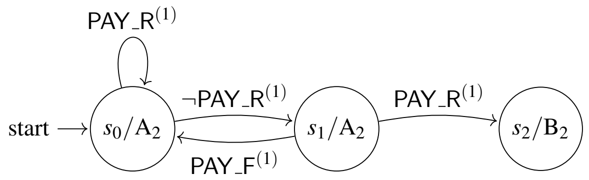
We introduce in the following an example of how to use Moore machines to capture two strategies that the landlord and the tenant should consider in the case of the motivating example.

Example 14 (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 4: 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. 4).

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

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 once both strategies are fixed and transformed to compute their outcome. To do so we introduce the product of two complementary Moore machines.

Definition 21 (Product of Complementary Determinstic Moore Machines)**.**

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

be two 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

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

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 15 (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 14 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, written $\{A_1, A_2\}$ 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.*
Concretely, substituting the outputs from Fig. 4, the product word begins as
$\{\mathsf{PAY\_R}^{(1)}, \mathsf{OCC}^{(2)}, \mathsf{PAY\_R}^{(2)}\}\ (\{\mathsf{OCC}^{(1)}, \mathsf{PAY\_R}^{(1)}, \mathsf{OCC}^{(2)}, \mathsf{PAY\_R}^{(2)}\})^{\omega}$.

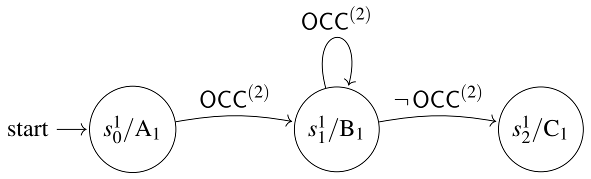
6 The Two-Agents Collaborative Normative Logic TACNL

6.1 Syntax of TACNL

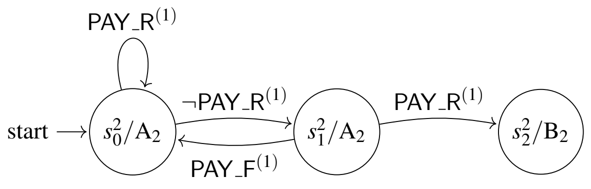
As summarized in Fig. 6, the syntax of TACNL is organized into three blocks: regular expressions, literals, and contracts.

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* $(re \mid re)$, *concatenation* $(re; re)$ for next-month sequencing, *power* re^n (exactly n repetitions), and *Kleene plus* re^+ (one or more repetitions). The *wildcard* $*$ is the union of all $A \subseteq \Sigma$ used to skip a position, and $\emptyset$ denotes the *empty language*. 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 $(*;)$, or "three consecutive months" $(re)^3$.

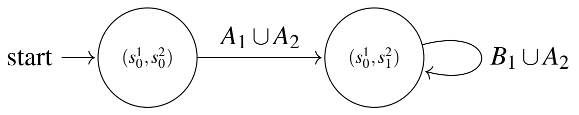
Literals (ℓ). This block provides the primitive deontic statements for a single month: *obligation* $\mathbf{O}_p(a)$, *prohibition* $\mathbf{F}_p(a)$, and *power* $\mathbf{P}_p(a)$, plus the constants *valid* $\top$ and *invalid* $\perp$. Here $p \in \{1, 2\}$ identifies the party (tenant or landlord)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



(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 5: 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 $\lambda_1(s_i^1) \cup \lambda_2(s_j^2)$.

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, conditions upon which they are activated or terminated, and how the clauses relate to each other regarding reparation or the timed order between them. 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:

6.2 Illustrating the encoding in TACNL for the motivating example

Example 16 (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. 6). We define the collaboration alphabet that captures all joint actions*

Figure 6: 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	$\mathsf{re} ::=$	A	[tagged action set]
	$\mid$	$*$	[trump card]
	$\mid$	ε	[empty word]
	$\mid$	$\emptyset$	[empty set of actions]
	$\mid$	$(\mathsf{re} \mid \mathsf{re})$	[union]
	$\mid$	$\mathsf{re}\ ;\ \mathsf{re}$	[concatenation]
	$\mid$	re^n	[n-repetition]
	$\mid$	re^+	[Kleene plus]
Literals	$\ell ::=$	$\mathbf{O}_p(a)$	[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]

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 action of requesting 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:*

$$C_5 := \lceil *^+ ; \{\mathsf{Notif\_T}^{(1)}\} ; *^3 \rceil (\ C_3 \ \wedge\ C_2 \ \wedge\ C_4\ ).$$

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 weither a contract was satisfied.

7 The notion of tight semantics:

7.1 The need of tight semantics

Contracts in TACNL are subject to *when* clauses are first met or first broken. A rent payment that occurs *now* settles the duty at this exact position; any later symbols should not re-open (nor re-justify) the same decision. Likewise, the first missed deadline fixes the violation point; subsequent steps are merely after-the-fact evidence. Before introducing the formal machinery, we motivate the need for a *tight*, prefix-based semantics that singles out these decisive instants.

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.

With this motivation in place, we now introduce the language-theoretic operators and automata constructions that realize these frontiers and, subsequently, define the tight five-valued semantics.

7.2 Regular Language-engineering for tight behavior

For the sake of simplicity this subsection shows how to build the motivated tight semantics on *any regular languages* defined over $L \subseteq \Sigma^*$. We introduce prefix-based operators that isolate the first decisive points of *acceptance* and *rejection*, and separate them from their *extension by any suffix*. The outcome is a disjoint and complete five-way partition of Σ^* tailored to tight semantics. We write $u \preceq v$ when u is a (possibly empty) prefix of v, and $u \prec v$ for a strict prefix. For $S, X \subseteq \Sigma^*$, $SX := \{uv \mid u \in S, v \in X\}$, and $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$.

Definition 22 (Prefix-closure and bad-prefix). *For a language $L \subseteq \Sigma^*$,*

$$\mathsf{PCl}(L) := \{u \in \Sigma^* \mid \exists v \in L : u \preceq v\}, \qquad \mathsf{BPl}(L) := \Sigma^* \setminus \mathsf{PCl}(L).$$

Definition 23 (Minimal elements). *For any $S \subseteq \Sigma^*$,*

$$\mathsf{Min}(S) := \{u \in S \mid \nexists v \in S : v \prec u\}.$$

In particular, $\mathsf{Min}(L)$ *are the* first acceptance *words, and* $\mathsf{Min}(\mathsf{BPl}(L))$ *are the* first rejection *(minimal bad) words.*

Definition 24 (Additional frontier definitions)**.**

$$\begin{array}{rcll} \mathsf{IAL}(L) & := & \mathsf{Min}(L)\,\Sigma^+ & \textit{(irrelevant acceptance)}, \\ \mathsf{SBad}(L) & := & \mathsf{Min}(\mathsf{BPI}(L)) \setminus \mathsf{IAL}(L) & \textit{(strict bad frontier)}, \\ \mathsf{IRL}(L) & := & \mathsf{SBad}(L)\,\Sigma^+ & \textit{(irrelevant rejection)}. \end{array}$$

Why the subtraction $\mathsf{Min}(\mathsf{BPI}(L)) \setminus \mathsf{IAL}(L)$**?** A word may be simultaneously a minimal bad prefix and a strict extension of a minimal acceptance (e.g. aa for $L = \{a\}$ over $\Sigma = \{a,b\}$). We resolve this tie *in favor of acceptance*, classifying such words as *acceptance overhead*. Removing $\mathsf{IAL}(L)$ from $\mathsf{Min}(\mathsf{BPI}(L))$ keeps the two frontiers disjoint.

Deconstructing regular language for tight behavior

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

1. $\mathsf{Min}(L) \subseteq \mathsf{PCl}(L)$, $\mathsf{Min}(\mathsf{BPI}(L)) \subseteq \mathsf{BPI}(L)$, *and* $\mathsf{Min}(L) \cap \mathsf{Min}(\mathsf{BPI}(L)) = \emptyset$.
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 $\mathsf{Min}(\mathsf{BPI}(L)) \subseteq \mathsf{BPI}(L)$ holds by definition of minimality within $\mathsf{BPI}(L)$. Disjointness follows because $\mathsf{PCl}(L) \cap \mathsf{BPI}(L) = \emptyset$.

(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. □

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* $L \subseteq \Sigma^*$*,*

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

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 $u \in \mathsf{BPI}(L)$ has a unique shortest bad prefix $b \in \mathsf{Min}(\mathsf{BPI}(L))$ 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. □

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

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

Five semantic regions

Definition 25 (Five semantic regions)**.** *For any $L \subseteq \Sigma^*$, define:*

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

Theorem 7.1 (Five-way partition of Σ^*)**.** *For every $L \subseteq \Sigma^*$, the space of all possible words could be decomposed into:*

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

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

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

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

Example 17 (Five regions illustration on a simple language)**.** *Let $\Sigma = \{a,b\}$ and $L = \{a\}$.*

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

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:

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

7.2.1 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:

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

Definition 26 (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 7.1, 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 7.2 (Frontier and evolution properties of the tight five semantics)**.** *Fix $L \subseteq \Sigma^*$ and $u \in \Sigma^*$. Then:*

- ***(A)*** ***Post-eager acceptance irrelevance.*** *If $[\![u \models L]\!]_5 = \top^t$, then for every strict longer word v with $u \prec v$ we have $[\![v \models L]\!]_5 = \top^p$.*
- ***(B)*** ***Unique eager-acceptance frontier before post-acceptance.*** *If $[\![u \models L]\!]_5 = \top^p$, then there exists a* unique *strict prefix $m \prec u$ such that $[\![m \models L]\!]_5 = \top^t$.*
- ***(C)*** ***Post-rejection irrelevance.*** *If $[\![u \models L]\!]_5 = \bot^t$, then for every strict longer word v with $u \prec v$ we have $[\![v \models L]\!]_5 = \bot^p$.*
- ***(D)*** ***Unique eager-rejection frontier before post-rejection.*** *If $[\![u \models L]\!]_5 = \bot^p$, then there exists a* unique *strict prefix $m \prec u$ such that $[\![m \models L]\!]_5 = \bot^t$.*
- ***(E)*** ***Two-way continuation (unique minimal frontiers).*** *If $[\![u \models L]\!]_5 = ?$, then there exist unique integers $\ell_{\top^t}, \ell_{\bot^t} \geq 1$ such that:*
  - *there exists a unique suffix $v \in \Sigma^{\ell_{\top^t}}$ with $[\![uv \models L]\!]_5 = \top^t$, and for every strict prefix v' of v we have $[\![uv' \models L]\!]_5 = ?$;*
  - *if some extension of u eventually leaves $\mathsf{PCI}(L)$, then there exists a unique suffix $x \in \Sigma^{\ell_{\bot^t}}$ with $[\![ux \models L]\!]_5 = \bot^t$, and for every strict prefix x' of x we have $[\![ux' \models L]\!]_5 = ?$.*

Proof. As in the indexed setting, replace the slice $w[i,j]$ by the unindexed prefix u. Key facts: $\mathsf{Min}(L)$ and $\mathsf{Min}(\mathsf{BPI}(L))$ are prefix-free; $\mathsf{IAL}(L) = \mathsf{Min}(L)\,\Sigma^+$; $\mathsf{IRL}(L) = \mathsf{SBad}(L)\,\Sigma^+$; and $\mathsf{ER}(L) = \mathsf{Min}(\mathsf{BPI}(L)) \setminus \mathsf{IAL}(L)$. 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. □

7.2.2 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 30.

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

Definition 27 (Five-region automata)**.** *Let $L \subseteq \Sigma^*$ be a regular language and let*

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

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

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

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

Definition 28 (Five-region Moore machine)**.** *Let $L \subseteq \Sigma^*$ be a regular language and let*

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

be the five DFAs from Definition 27, 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*

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

where

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

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

Example 18 (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* $\mathsf{Pre}(L), \mathsf{EA}(L), \mathsf{ER}(L), \mathsf{IAL}(L), \mathsf{IRL}(L)$.

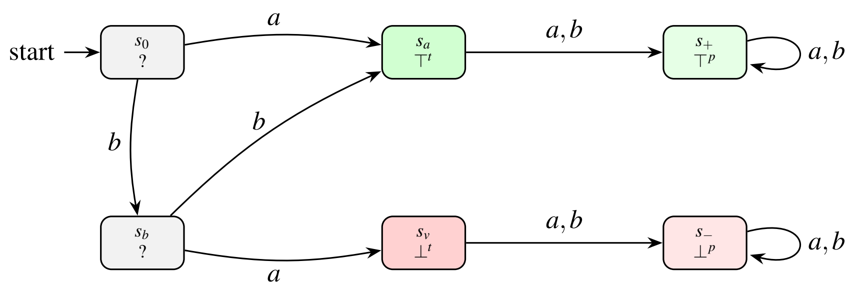


Figure 7: Minimized five-valued Moore machine for $L = \{a, ab, bb\}$. Each node shows its name and output verdict ($\mathbb{V}_5$). Green states denote satisfaction, red denote violation, gray undecided. From s_0, input a or bb yields $\top^t$, input ba yields $\bot^t$, and further inputs move to the post-frontier verdicts $\top^p$ or $\bot^p$.

Lemma 8 (Correctness of the tight-product Moore machine)**.** *Let $L \subseteq \Sigma^*$ be regular and let $M_{5tight}(L)$ be as in Definition 28. 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 26.

Proof. Let the five DFAs be

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

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

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

For each component DFA,

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

By Theorem 7.1, 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 28) returns the unique verdict of Definition 26. Therefore $\lambda(\delta^*(s^0,u)) = [\![u \vDash L]\!]_5$. □

Proposition 7.3 (Linear-size Moore machine)**.** *Let $\mathscr{A}(L) = (Q,\Sigma,\delta,q_0,F)$ be a completed DFA for L, and let $M_{5tight}(L)$ be the tight five-valued Moore machine of Definition 28. Then the number of reachable states of $M_{5tight}(L)$ is linear in $|Q|$:*

$$\left|S_{\text{reach}}(M_{5tight}(L))\right| \;=\; \left|\mathsf{Reach}(\mathscr{A}(L))\right| \;\le\; |Q|.$$

Proof. We argue directly from the constructions of the five DFAs used in $M_{\text{5tight}}(L)$.

(Same transition graph). Starting from a completed DFA $\mathscr{A}(L) = (Q,\Sigma,\delta,q_0,F)$:

- The prefix-closure automaton $\mathsf{PC}(\mathscr{A}(L)) = (Q,\Sigma,\delta,q_0,\mathsf{Live})$ changes only the accepting set to Live (backward reachability to F).
- Its complement for bad-prefixes $\mathsf{BP}(\mathscr{A}(L)) = (Q,\Sigma,\delta,q_0,Q\setminus\mathsf{Live})$ flips finals, keeping (Q,Σ,δ).
- The minimal-frontier automaton $\mathsf{MinAut}(\mathscr{A}(L)) = (Q,\Sigma,\delta,q_0,F_{\min})$ 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).
- $\mathsf{ER}(L) = \mathsf{Min}(\mathsf{BPI}(L))\setminus\mathsf{IAL}(L)$ is realized on the same (Q,Σ,δ,q_0) by taking $F_{\text{ER}} := F_{\min(\text{Bad})}\setminus F_{\text{IAL}}$, 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 $M_{\text{5tight}}(L)$, the product state reached by u is necessarily diagonal: (q,q,q,q,q).

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

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

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

$$\left|S_{\text{reach}}(\mathscr{M}_{\text{5tight}}(L))\right| \;=\; \left|\mathsf{Reach}(\mathscr{A}(L))\right| \;\le\; |Q|.$$

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

Additionally, each component DFA is deterministic (and can be completed), so $\mathscr{M}_{\text{5tight}}(L)$ 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 7.2: 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 $\mathscr{A}(L)$ we systematically build the prefix-closure automaton $\mathsf{PC}(\mathscr{A}(L))$, which recognizes $\mathsf{PCl}(L)$; its complement $\mathsf{BP}(\mathscr{A}(L))$, which recognizes $\mathsf{BPl}(L)$; and the minimal-frontier automaton $\mathsf{MinAut}(\mathscr{A}(L))$, 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 $\mathsf{ER}(L) := \mathsf{Min}(\mathsf{BPl}(L)) \setminus \mathsf{IAL}(L)$.

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*.

7.3 Illustration through Regular expressions from TACNL

We demonstrate how we can define monitors for regular expressions from TACNL by first defining the language semantics of the regular expression, then via a traditional transformation a deterministic automaton. Finally we use the transformation shown in the previous subsection in order to define the 5 tight semantic monitor for the formula. This subsection fixes the semantics of the regular expressions that act as temporal guards in TACNL .

7.3.1 Semantics for regular expression

We work over the *letter alphabet*

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

Definition 29 (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:*

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

We write $\mathscr{L}(\mathsf{re}) := \{\pi \in \Gamma^* \mid \pi \models_{\mathsf{re}} \mathsf{re}\}$ *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** $*$**.** 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** $(re_1 \,; re_2)$**.** 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 $\pi \models_{\mathsf{re}} re \,; re^{n-1}$.
- **Kleene plus** re^+**.** Some positive iteration holds: $\exists n \ge 1$ with $\pi \models_{\mathsf{re}} re^n$.

All satisfaction is defined on *finite* traces, we show how to detect trigger and terminating condition on regular expression using the 5 valued semantics.

7.3.2 Automata construction matching the denotational semantics

We now give a concrete, standard pipeline that realizes the semantics of Definition 29 *exactly* by an automaton over the alphabet $\Gamma = 2^{\Sigma_C^{(1)} \cup \Sigma_C^{(2)}}$.

Stage 1 Thompson-style ε-NFA $\mathscr{A}_\varepsilon(re)$ (alphabet-aware). We build $\mathscr{A}_\varepsilon(re)$ by structural recursion on re, using the usual two distinguished states $(s_{\text{in}}, s_{\text{out}})$ per fragment and ε-transitions for wiring ([28, 18, 27]). The only twist is how we treat letters, since an atom A matches *any* Γ-letter X that *covers* A (Definition 29).

- **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 $*$:** 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 $(re_1\,; re_2)$:** 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 $\mathscr{L}(re)$.

Stage 2 Determinization. Apply the standard subset construction with ε-closures to obtain a DFA $\mathscr{A}_D(re) = (Q, \Gamma, \delta, q_0, F)$ that recognizes the same language ([26, 18]).

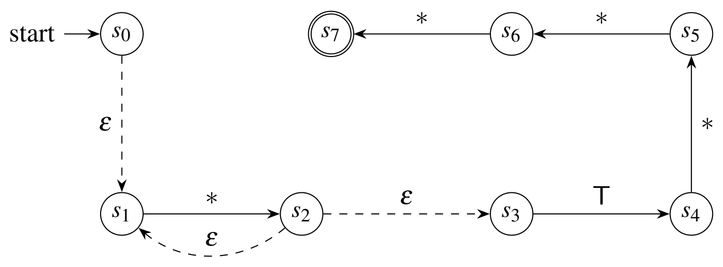
Correctness (sketch). By structural induction on *re*. The letter fragments implement exactly the one-step clauses (A, $*$, $\emptyset$), union and sequencing are the usual ε-wiring proofs, and re^n (unrolled) and re^+ (loop back from the exit) match the inductive clauses in Definition 29. Determinization and completion preserve language.

From $\mathscr{A}(re)$ to the tight layers. Follow the same steps from Definition .28:

Example 19 (End-to-end construction). *Continuing from Example. 16, we demonstrate the automata construction for C_5:*

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

Figure 8 depicts the Thompson-style ε-NFA for re_{C_5}, and Figure 9 shows its determinized and completed DFA.



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

Figure 8: Thompson-style ε-NFA for $\mathsf{re}_{C_5} = *^+ \ ; \ \{\mathsf{Notif\_T}^{(1)}\} \ ; \ *^3$ over $\Gamma = 2^\Sigma$. From s_0 we enter the $*^+$ 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 29.

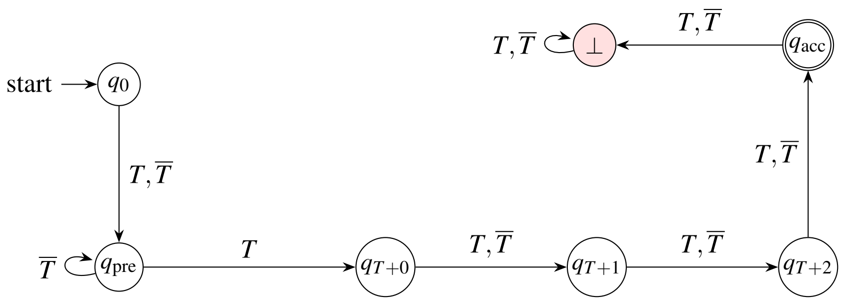


Figure 9: Determinized and completed DFA for $\mathsf{re}_{C_5} = *^+;\{\mathsf{Notif\_T}^{(1)}\};*^3$, abstracting Γ by $T := \{A \in \Gamma \mid \mathsf{Notif\_T}^{(1)} \in A\}$ and $\overline{T} := \Gamma \setminus T$. State q_{pre} collects the initial $*^+$ segment; transition on T begins the "+3 letters" counter ($q_{T+0} \to q_{T+1} \to q_{T+2} \to q_{\text{acc}}$). Any overrun moves to the sink.

7.3.3 From Language automaton to Tight monitor construction

Definition 30 (Tight monitor construction for regular expressions)**.** *Let* re *be a regular expression over the alphabet* Γ *and let* $L := \mathscr{L}(\mathsf{re}) \subseteq \Gamma^*$ *be its language as in Definition 29. Let*

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

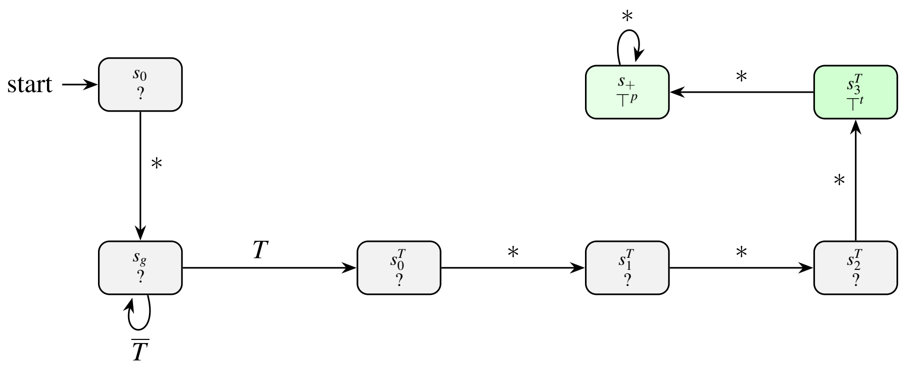
be the five-region Moore machine for L from Definition 28. The tight satisfaction monitor *for* re *is this Moore machine:*

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

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:

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

Example 20 (Tight monitor for C_5)**.** *Continuing Example 19, Figure 10 shows the compact five-valued Moore monitor obtained by applying the tight monitor construction of Definition 30 to the regular expression* re_{C_5}.



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

Figure 10: Compact five-valued Moore monitor for $re_{C_5} = *^+;\{\mathsf{Notif\_T}^{(1)}\};*^3$. 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.

8 Tight Forward Reasoning on Contract Compliance

8.1 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):

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

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

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

Definition 31 (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 judgements 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.

8.1.1 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 32 (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 $\langle A\rangle$			***Tight violation on $\langle A\rangle$***		
$\langle A\rangle \models_{\top^t} \top$	$\overset{\text{def}}{=}$	*true*.	$\langle A\rangle \models_{\bot^t} \top$	$\overset{\text{def}}{=}$	*false*.
$\langle A\rangle \models_{\top^t} \bot$	$\overset{\text{def}}{=}$	*false*.	$\langle A\rangle \models_{\bot^t} \bot$	$\overset{\text{def}}{=}$	*true*.
$\langle A\rangle \models_{\top^t} \mathbf{O}_1(a)$	$\overset{\text{def}}{=}$	$\{a^{(1)}, a^{(2)}\} \subseteq A$.	$\langle A\rangle \models_{\bot^t} \mathbf{O}_1(a)$	$\overset{\text{def}}{=}$	$\{a^{(1)}, a^{(2)}\} \nsubseteq A$.
$\langle A\rangle \models_{\top^t} \mathbf{F}_1(a)$	$\overset{\text{def}}{=}$	$\{a^{(1)}, a^{(2)}\} \nsubseteq A$.	$\langle A\rangle \models_{\bot^t} \mathbf{F}_1(a)$	$\overset{\text{def}}{=}$	$\{a^{(1)}, a^{(2)}\} \subseteq A$.
$\langle A\rangle \models_{\top^t} \mathbf{P}_1(a)$	$\overset{\text{def}}{=}$	$(a^{(1)} \in A) \Rightarrow (a^{(2)} \in A)$.	$\langle A\rangle \models_{\bot^t} \mathbf{P}_1(a)$	$\overset{\text{def}}{=}$	$(a^{(1)} \in A) \wedge (a^{(2)} \notin A)$.

The case $p = 2$ is symmetric.

Example 21 (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

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

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 decide at the first letter, the overall collapsed verdict follows from $\langle A\rangle$:*

$$\langle A, A'\rangle \models_{\top^p} \mathbf{O}_1(a),$$
$$\langle A, A'\rangle \models_{\top^p} \mathbf{P}_1(a),$$
$$\langle A, A'\rangle \models_{\perp^p} \mathbf{P}_2(b).$$

8.1.2 Binary Contract Operators Tight Semantics

Binary contract operators combine two contracts into structured compositions that capture parallel, sequential, or conditional behaviour. 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 requires, on the other hand, that either C succeeds directly, or at the first tight violation of C the repair C' must succeed on the remainder.

Definition 33 (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 $(C \wedge C')$

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

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

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

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

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

Reparation $(C \blacktriangleright C')$

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

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

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 22 (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 (longest prefix). *Consider the trace*

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

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

Tight satisfaction (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*

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

At t=0, $\langle A_0 \rangle \models_{\top^t} C_2$ and $\langle A_0 \rangle \models_{\bot^t} \mathbf{O}_1(\mathsf{PAY\_R})$, 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, $\pi_{viol} \models_{\bot^t} C_3$ (as $\pi[0,0] \models_{\bot^t} \mathbf{O}_1(\mathsf{PAY\_R})$ and $\pi[1,1] \models_{\bot^t} \mathbf{O}_1(\mathsf{PAY\_F})$). Thus, the overall violation arises from C_3, and by conjunction, $\pi_{viol} \models_{\bot^t} (C_2 \wedge C_3)$.

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

8.1.3 Repetition Contracts Tight Semantics

Definition 34 (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:*

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

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 obligations, 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 unfoldings 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.

8.1.4 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 35 (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 $(\pi \models_{?} X$ or $\pi \models_{\top^t} X)$.

Example 23 (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: $\pi_{\mathsf{viol}} \models_{\bot^t} C_4$.

(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.

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

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

Tight violation.

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

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

8.1.5 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. 32, a literal is decided on a single letter: $\langle A \rangle \models_{\top^t} \ell$ iff the letter constraint holds, and $\langle A \rangle \models_{\bot^t} \ell$ 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:

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

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. 33, 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 $\pi[0,j] \models_{\bot^t} C$ nor $\pi[0,j] \models_{\bot^t} C'$ holds. Since a tight violation of a conjunct 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. 33, 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 $j \leq k$, (IH-C-sat) forbids $\pi[0,j] \models_{\bot^t} C$; for any $j > k$, (IH-C'-sat) applied to the suffix forbids $\models_{\bot^t} C'$ 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 by (IH-C-viol) and (IH-C'-viol).

Reparation $C \blacktriangleright C'$. By Def. 33, 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) 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. 35, 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. 35, to derive the two implications.

All constructors preserve the two "no-backtrack" properties, hence the claim for all C. □

Theorem 8.1 (Consistency of the forward looking 5 tight semantics)**.** *The five forward satisfaction relations* $\{\models_?, \models_{\top^t}, \models_{\perp^t}, \models_{\top^p}, \models_{\perp^p}\}$ *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 $\pi \models_{\perp^t} C$ 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 $\pi[0,k'] \models_{\perp^t} C$, and these cannot coexist.

By Definition 31, 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 $\models_{\perp^t}$), or strictly follows the unique frontier ($\models_{\top^p}$ or $\models_{\perp^p}$). 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, $\{\models_?, \models_{\top^t}, \models_{\perp^t}, \models_{\top^p}, \models_{\perp^p}\}$ forms a partition of prefixes, proving the claim. □

8.2 Monitor Construction for Tight Contract Satisfaction

Definition 36 (Tight satisfiaction monitor)**.** *The* tight satisfaction monitor*, written* $\mathscr{M}^{TS}$*, is a Moore machine whose output alphabet is the five-valued verdict set* $\mathbb{V}_5$*. Formally,*

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

where:

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

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

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. 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 30.

Definition 37 (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 30.*

8.2.1 Construction for Literal Contracts

From Tight Semantics to 5-Valued Monitoring. The literal clauses above define one-step satisfaction and violation judgments 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 38 (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:*

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

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

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 32 but refines it with explicit prefix continuity.

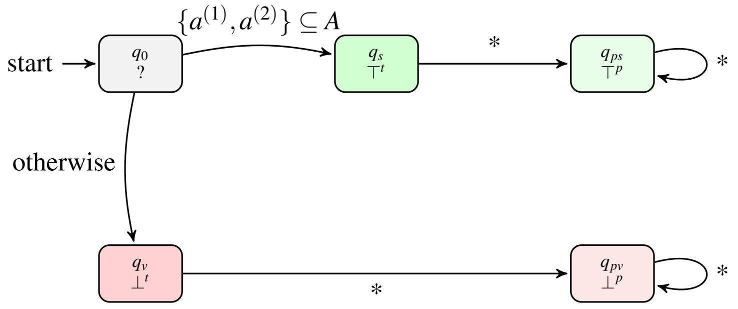


Figure 11: Compact 5-verdict Moore machine for the obligation literal $\mathbf{O}_1(a)$. The symbol $*$ stand for any $A \subseteq 2^{\Sigma}$. Each node displays its internal state and the corresponding output verdict $\in \mathbb{V}_5$. The first joint execution of $a^{(1)}$ and $a^{(2)}$ yields $\top^t$, otherwise $\bot^t$; subsequent steps emit the post-frontier verdicts $\top^p$ or $\bot^p$.

8.2.2 Construction for Binary Contract Operators

For binary contract operators of the form $C \ \mathsf{op}\ C'$ with $\mathsf{op} \in \{\wedge, \ ; \ , \blacktriangleright\}$, 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 39 (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 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 39. 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 $\perp^p$ immediately.
- A tight violation in one component produces $\perp^t$ 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 implements the tight semantics of conjunction correctly. □

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 component monitors and redirects transitions that correspond to 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 40 (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 output the same verdict according to the sequence semantics:*

$$\lambda_5^;\big(\delta_;(q_0^;, \pi)\big) = \llbracket \pi \vDash C\,;\,C' \rrbracket_5.$$

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 41 (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

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

- *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*
$$\delta_{\blacktriangleright}(q,A) = \begin{cases} q_0' & \textit{if } q \in Q \textit{ and } \lambda_5(\delta(q,A)) = \bot^t, \\ \delta(q,A) & \textit{if } q \in Q \textit{ and } \lambda_5(\delta(q,A)) \notin \{\bot^t, \bot^p\}, \\ \delta'(q,A) & \textit{if } q \in Q'. \end{cases}$$

- *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 the transitions that lead to a tight violation of C rather than 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 remainder of the 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 41. 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 40 and 41 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 RV literature [10, 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 [14], where a secondary clause is activated after the primary obligation fails. Hence, the constructions in Definitions 40 and 41 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 24 (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 12. The Letter classes used on edges correspond to shorthand for literal satisfaction or violation conditions:

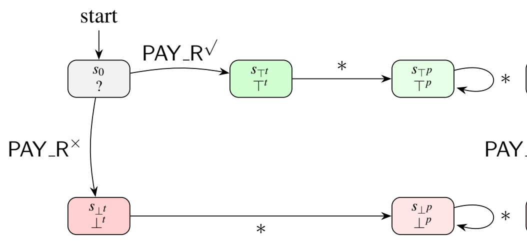
$$\mathsf{PAY\_R}^{\surd} := \{A \in \Gamma \mid \{\mathsf{PAY\_R}^{(1)}, \mathsf{PAY\_R}^{(2)}\} \subseteq A\}, \qquad \mathsf{PAY\_R}^{\times} := \Gamma \setminus \mathsf{PAY\_R}^{\surd},$$

$$\mathsf{PAY\_F}^{\surd} := \{A \in \Gamma \mid \{\mathsf{PAY\_F}^{(1)}, \mathsf{PAY\_F}^{(2)}\} \subseteq A\}, \qquad \mathsf{PAY\_F}^{\times} := \Gamma \setminus \mathsf{PAY\_F}^{\surd},$$

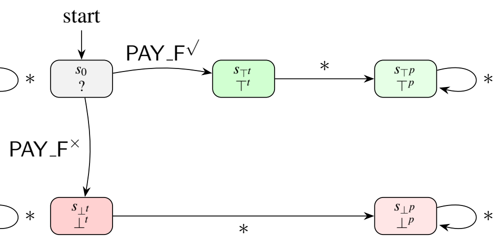
$$\mathsf{OCC}^{\surd} := \{A \in \Gamma \mid \mathsf{OCC}^{(2)} \in A \Rightarrow \mathsf{OCC}^{(1)} \in A\}, \qquad \mathsf{OCC}^{\times} := \Gamma \setminus \mathsf{OCC}^{\surd}.$$

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.

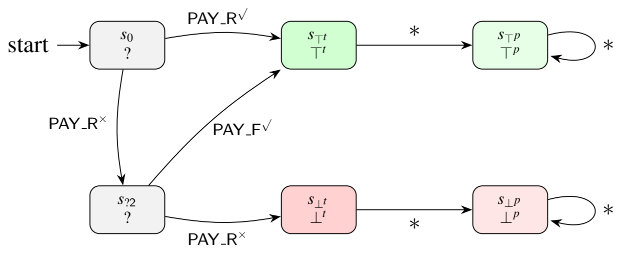
Reading Fig. 12d with the letter classes defined above: from s_0 (pre) there are three behaviors. (i) If $\mathsf{OCC}^{\surd} \wedge \mathsf{PAY\_R}^{\surd}$ holds in the current letter, both conjuncts succeed (perm is respected and the primary obligation is met), so the product emits $\top^t$ and moves 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 $\mathsf{OCC}^{\surd} \wedge \mathsf{PAY\_R}^{\times}$ holds, the reparation branch of C_3 is activated, and we move to the waiting state s_1 (still $?$). From s_1, $\mathsf{PAY\_F}^{\surd}$ 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.*



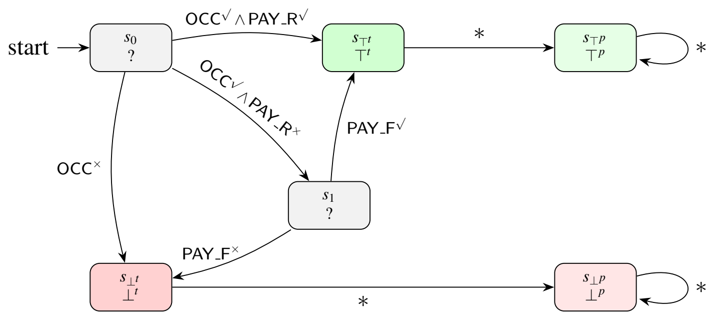
(a) $\mathbf{O}_1(\mathsf{PAY\_R})$. Satisfaction occurs when both agents perform $\mathsf{PAY\_R}$.



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



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



start
s_0 ?
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\checkmark}$
$s_{\top^t}$ $\top^t$
$*$
$s_{\top^p}$ $\top^p$
$*$
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\times}$
$\mathsf{PAY\_F}^{\checkmark}$
$\mathsf{OCC}^{\times}$
s_1 ?
$\mathsf{PAY\_F}^{\times}$
$s_{\bot^t}$ $\bot^t$
$*$
$s_{\bot^p}$ $\bot^p$
$*$

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

Figure 12: 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. $\mathsf{OCC}^{\checkmark} := \{A \mid \{\mathsf{OCC}^1\} \not\subset A$ 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 $\mathsf{PAY\_R}^{\checkmark}$.

8.2.3 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 42 (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*** $(q \in Q_r^{open})$***.*** *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 guard impossible (enter $\bot^t$ or $\bot^p$) to a vacuous-success sink that emits $\top^t$ once then $\top^p$. 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 42 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 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 is either simulating the guard with the correct decisive redirection points, or simulating 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 43 (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 43 for guard[re]C. Then, for every finite trace π,

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

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 43. 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 $[\![\varepsilon \vDash \text{guard}[re]C]\!]_5$.

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

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

- **Guard impossible** $\lambda_5^r(x) \in \{\bot^t, \bot^p\}$. This corresponds exactly to $\pi \ \vDash_{\bot^t} \ re$. 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:

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

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

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

for all traces π. □

8.2.4 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 $\textbf{Rep}(C)$, we obtain an infinite cycle without ever reporting tight success.

The following constructions make these ideas explicit.

Definition 44 (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*

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

Disjoint copies. *For each i* $\in \{1,\dots,n\}$*, 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.

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

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

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

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

For the last copy $i = n$:

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

Sink states absorb:

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

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

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

For sink states:

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

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

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

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

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

Definition 45 (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 **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 **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 **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 25 (Open ended contract monitor construction)**.** *W*

Example 26 (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 13.*

(a) $\mathsf{TSMC}(\mathbf{Rep}(C_3))$ ***(Subfigure 13a).*** *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 45: 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 13b).*** *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 13c).*** *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:*

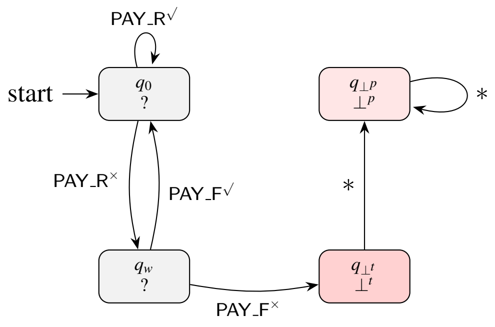
$$
\begin{aligned}
&s_0 \times q_0 \xrightarrow{\mathsf{PAY\_R}^{\times}} s_1 \times q_w,\\
&s_1 \times q_w \xrightarrow{\mathsf{PAY\_F}^{\times}} (s_0, s_1) \times q_{\perp^t},\\
&s_1 \times q_w \xrightarrow{\mathsf{PAY\_F}^{\checkmark} \wedge \overline{T}} s_1 \times q_0,\\
&s_1 \times q_0 \xrightarrow{\mathsf{PAY\_R}^{\times}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

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.*

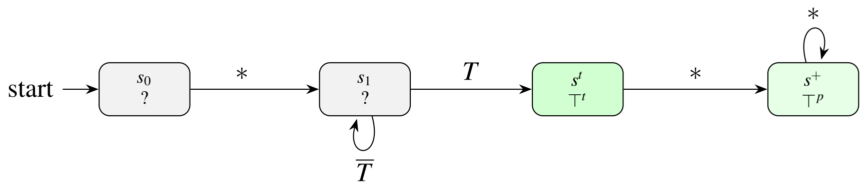
Acceptance region (right). *When the guard fires on a letter in T, the product moves to $s^t \times (q_0, q_w)$ with verdict $\top^t$ provided the repetition contract has not violated. From there, any continuation leads to the post-acceptance state $s^+ \times Q$ that loops on all letters and emits $\top^p$.*

This behavior follows directly from Definition 43: 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

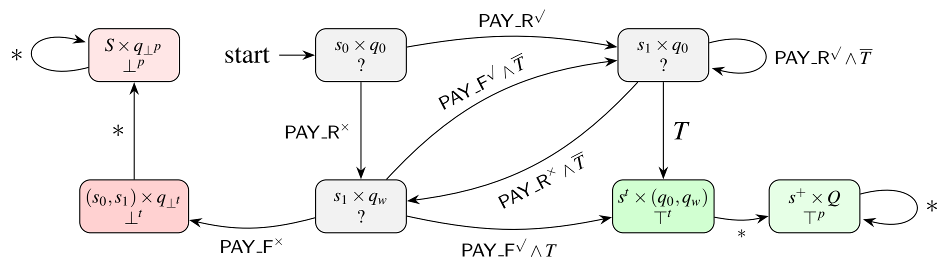


(a) Monitor for $\mathbf{Rep}(C_3)$.



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

(b) Monitor for the regular expression $*^+\ ;\ \{terminate^{(1)}\}$.



(c) Tight satisfaction monitor construction for $\lceil *^+\ ;\ \{terminate^{(1)}\}\rceil \mathbf{Rep}(C_3)$.

Figure 13: Progressive construction for $\lceil (*^+;\mathsf{Notif\_T}^{(1)})\rceil \mathbf{Rep}(C_3)$.

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 .

8.3 Forward-Looking Blaming Semantics

The tight semantics of TACNL identifies when a contract is satisfied or violated but does not explain *who* caused a violation. To attribute responsibility, we refine the violation verdicts according to which party failed to meet the corresponding normative requirement. We break down tight and post violations into violations caused by agent 1, violations caused by agent 2, violations caused jointly by both, and violations 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.

8.3.1 Blame Rules for Literals

Definition 46 (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 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:*

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

Units: Valid ($\top$) and invalid ($\bot$). *$\neg(\langle A\rangle \vDash_{\bot^t_S} \top)$ for all S. $\langle A\rangle \vDash_{\bot^t_0} \bot$ 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":*

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

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 $\bot$), never $\{12\}$.*

Example 27 (Obligation, prohibition, and power blame)**.** *By fixing $p = 1$, $\overline{p} = 2$. Consider the following letters $A \in \Gamma$:*

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

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

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

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

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

$A = \{a^{(1)}\}$:	$\langle A\rangle \vDash_{\bot^t_2} \mathbf{P}_1(a)$	*(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).*

8.3.2 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:

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

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 that results from the

forward-looking blame. The blame goes to the agent responsible for the first violation of the contract: so either C or C', but both contract could be violated on the same time point, in this case the agent or agents responsible for *both simultaneous* violation gets 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:

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

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:

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

Intuition. A reparation clause becomes active only after a violation of C. The global blame set S therefore corresponds to the agents responsible for the violation of 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 28 (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:* $\pi_1 \vDash_{\bot^t_1} \mathbf{O}_1(\mathsf{PAY\_R})$.
- *The landlord (2) gets blamed for violating the power of the tenant to occupy the flat:* $\pi_1 \vDash_{\bot^t_2} \mathbf{P}_1(\mathsf{PAY\_R})$.

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:*
    $\pi \vDash_{\perp_2^t} \mathbf{O}_1(\mathsf{PAY\_R}) \blacktriangleright \mathbf{O}_1(\mathsf{PAY\_F})$.

8.4 From Tight Contract Satisfaction Monitor to tight Blame Monitor

Definition 47 (Blame monitor)**.** *The* blame monitor*, written $\mathscr{M}_{11}$, is a Moore machine whose output alphabet is the eleven-valued blame verdict set $\mathbb{V}_{11}$. Formally,*

$$\mathscr{M}_{11} = (Q, q_0, \Gamma, \mathbb{V}_{11}, \delta, \lambda_{11}),$$

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 48 (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 reuse of the tight satisfaction monitor construction of Definition 37. 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 reaches 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 judgements 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 49 (Blame refinement of a contract monitor). *Let $\mathscr{M}(C) = (Q, \Gamma, \delta, q_0, \lambda)$ 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 *$\mathscr{BM}(C)$ is constructed by retaining the control structure of $\mathscr{M}(C)$ while refining the violation outputs to pinpoint responsibility. Formally:*

$$\mathscr{BM}(C) = (Q, \Gamma, \delta, q_0, \lambda^{\mathscr{BM}}),$$

where the new output function $\lambda^{\mathscr{BM}} : Q \to \mathbb{V}_{11}$ is defined for every state $q \in Q$ (reached by some trace π) as follows:

$$\lambda^{\mathscr{BM}}(q) = \begin{cases} \lambda(q) & \text{if } \lambda(q) \in \{?,\ \top^t,\ \top^p\}, \\ \bot^t_S & \text{if } \lambda(q) = \bot^t \text{ and } \pi \vDash_{\bot^t_S} C, \\ \bot^p_S & \text{if } \lambda(q) = \bot^p \text{ and } \pi \vDash_{\bot^p_S} C. \end{cases}$$

Theorem 8.2 (Correctness and Consistency of the Blame Monitor). *Let C be a contract in* TACNL . *Let $\mathscr{M}^{TS}(C)$ be its tight satisfaction monitor with output function λ_5, and let $\mathscr{BM}(C)$ be its blame monitor with output function $\lambda^{\mathscr{BM}}$ (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.8.1).*

For every finite trace π, the following equality holds:

$$\lambda^{\mathscr{BM}}\big(\delta^{\mathscr{BM}}(q_0, \pi)\big) \; = \; \mathsf{Blame}(C, \pi).$$

Furthermore, the blame monitor is consistent with the tight satisfaction monitor for non-violating verdicts. For all traces π:

$$\mathsf{Blame}(C, \pi) \in \{?, \top^t, \top^p\} \implies \lambda^{\mathscr{BM}}\big(\delta^{\mathscr{BM}}(q_0, \pi)\big) \; = \; \lambda_5\big(\delta^{\mathscr{M}^{TS}}(q_0, \pi)\big).$$

Proof. The proof proceeds by structural induction on C. We define $\lambda^{\mathscr{BM}}$ (denoted λ_{11}) using λ_5 and verify both correctness and consistency for each operator.

Base Case: Literals (ℓ). We defined $\lambda^{lit}_{11}(q, A)$ such that if $\lambda_5(q) \in \{?, \top^t, \top^p\}$, then $\lambda^{lit}_{11}(q, A) = \lambda_5(q)$. 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 49.

Inductive Step: Conjunction ($C_1 \wedge C_2$). Let $q = (q_1, q_2)$. The definition of $\lambda^{\wedge}_{11}$ defaults to the combination table $\lambda^{\mathsf{comb}}_5$ whenever neither component outputs a blame verdict (which corresponds to neither component outputting $\bot^t$ or $\bot^p$).

$$\lambda^{\wedge}_{11}(q) = \lambda^{\mathsf{comb}}_5(\lambda_5(q_1), \lambda_5(q_2)) \quad \text{if no blame detected.}$$

Since $\lambda^{\wedge}_5$ 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. $\lambda^{\mathscr{BM}}(q) = \top^t \iff \lambda_5(q) = \top^t$ (and similarly for $\top^p, ?$).

Inductive Step: Sequence ($C_1;C_2$). The state space is partitioned into Q_1 and Q_2.

- If $q \in Q_1$: $\lambda_{11}^{;}(q) = \lambda_{11}^{1}(q)$. By IH, λ_{11}^{1} is consistent with λ_5^1. Since $\lambda_5^{;}(q) = \lambda_5^1(q)$ here, consistency is preserved.
- If $q \in Q_2$: $\lambda_{11}^{;}(q) = \lambda_{11}^{2}(q)$. By IH, λ_{11}^{2} is consistent with λ_5^2. Since $\lambda_5^{;}(q) = \lambda_5^2(q)$ 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 $\lambda_{11}^{\blacktriangleright}(q) = \lambda_5^1(q)$. Since $\lambda_5^{\blacktriangleright}(q) = \lambda_5^1(q)$, they are identical.
- If $q \in Q_2$: The primary contract failed. We defined $\lambda_{11}^{\blacktriangleright}(q) = \lambda_{11}^{2}(q)$. By IH, this is consistent with $\lambda_5^2(q)$. Since $\lambda_5^{\blacktriangleright}(q) = \lambda_5^2(q)$, consistency is preserved.

Conclusion. For all constructions, $\lambda^{\mathscr{BM}}(q) = \lambda_5(q)$ whenever $\lambda_5(q)$ is a non-violating verdict. Whenever $\lambda_5(q)$ is a violation ($\bot^t, \bot^p$), $\lambda^{\mathscr{BM}}(q)$ refines it to a blame verdict ($\bot_S^t, \bot_S^p$). 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:* $\{q \mid \lambda(q) = \bot^t\} = \bigcup_S \{q \mid \lambda^{\mathscr{BM}}(q) = \bot_S^t\}$,
- *The post violation states are split:* $\{q \mid \lambda(q) = \bot^p\} = \bigcup_S \{q \mid \lambda^{\mathscr{BM}}(q) = \bot_S^p\}$.

Proof of Theorem 8.2: 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 $\mathscr{BM}(C)$ 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 $\lambda_{11}^{lit}(q,A)$ as:

$$\lambda_{11}^{lit}(q,A) = \begin{cases} \bot_p^t & \text{if } \lambda_5(q) = \bot^t \text{ and } a^{(p)} \notin A, \\ & \text{(Subject failed to attempt)} \\ \bot_{\bar{p}}^t & \text{if } \lambda_5(q) = \bot^t \text{ and } a^{(p)} \in A \wedge a^{(\bar{p})} \notin A, \\ & \text{(Counterparty withheld cooperation)} \\ \bot_S^p & \text{if } \lambda_5(q) = \bot^p \text{ (inherits previous blame } S\text{)}, \\ \lambda_5(q) & \text{if } \lambda_5(q) \in \{?, \top^t, \top^p\}. \end{cases}$$

This matches the blame assignment for literals defined in Definition 49.

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.

$$\lambda_{11}^{\wedge}(q_1,q_2) = \begin{cases} \perp_{S_1 \cup S_2}^{t} & \text{if } \lambda_{11}^1(q_1) = \perp_{S_1}^{t} \text{ and } \lambda_{11}^2(q_2) = \perp_{S_2}^{t}, \\ \perp_{S_1}^{t} & \text{if } \lambda_{11}^1(q_1) = \perp_{S_1}^{t} \text{ and } \lambda_5^2(q_2) \in \{?, \top^t, \top^p\}, \\ \perp_{S_2}^{t} & \text{if } \lambda_{11}^2(q_2) = \perp_{S_2}^{t} \text{ and } \lambda_5^1(q_1) \in \{?, \top^t, \top^p\}, \\ \perp_{S_1 \cup S_2}^{p} & \text{if } \lambda_{11}^1(q_1) = \perp_{S_1}^{p} \text{ or } \lambda_{11}^2(q_2) = \perp_{S_2}^{p}, \\ \lambda_5^{\mathsf{comb}}(\lambda_5^1(q_1), \lambda_5^2(q_2)) & \text{otherwise.} \end{cases}$$

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. $\square$

In all cases, λ_{11} correctly maps the state to the specific blame verdict defined by the forward-looking semantics. We move now to illustrate this refinement on two interesting examples

Example 29 (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. 12d. 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

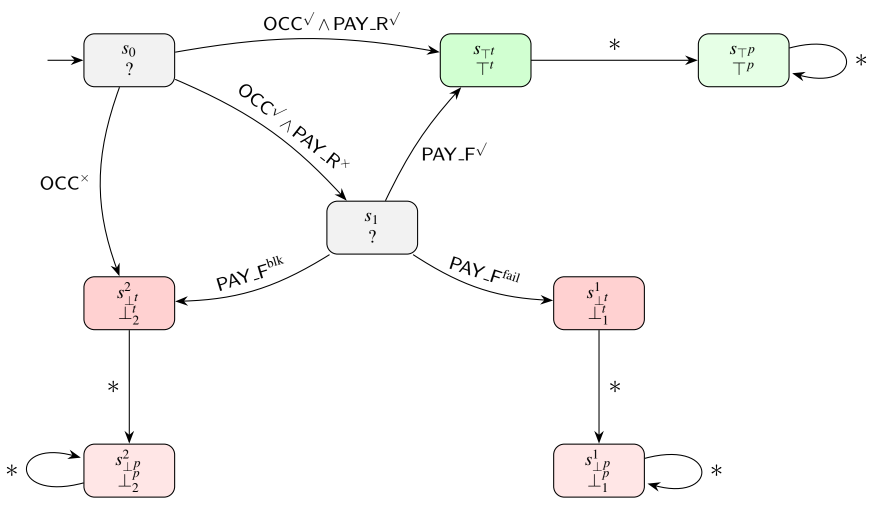

s_0 ?
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\checkmark}$
$s_{\top^t}$ $\top^t$
$*$
$s_{\top^p}$ $\top^p$
$*$
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\times}$
$\mathsf{PAY\_F}^{\checkmark}$
$\mathsf{OCC}^{\times}$
s_1 ?
$\mathsf{PAY\_F}^{\text{blk}}$
$\mathsf{PAY\_F}^{\text{fail}}$
$s^2_{\perp^t}$ $\perp^t_2$
$s^1_{\perp^t}$ $\perp^t_1$
$*$
$*$
$*$
$s^2_{\perp^p}$ $\perp^p_2$
$s^1_{\perp^p}$ $\perp^p_1$
$*$

Figure 14: Blame Monitor $\mathscr{BM}(C_2 \wedge C_3)$. **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. $\mathsf{PAY\_F}^{\text{fail}}$: Tenant does not attempt $\mathsf{PAY\_F}$ ($\mathsf{PAY\_F}^{(1)} \notin A$). $\mathsf{PAY\_F}^{\text{blk}}$: Tenant attempts $\mathsf{PAY\_F}$, Landlord blocks.

reparation (outputting ?) rather than emitting an immediate violation verdict. Consequently, 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 30 (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 31 (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.*

8.5 Conclusion and Limitation

We have presented a forward-looking semantics and a corresponding monitor construction that refines the standard satisfaction verdicts with responsibility assignments. 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.

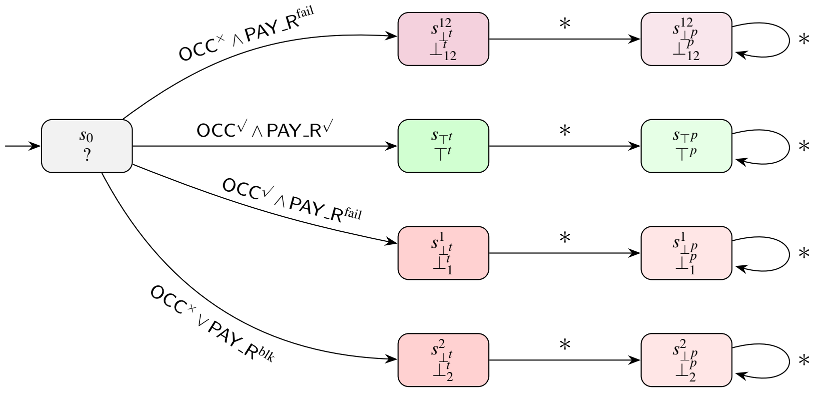

s_0
?
$\mathsf{OCC}^{\times} \wedge \mathsf{PAY\_R}^{\text{fail}}$
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\checkmark}$
$\mathsf{OCC}^{\checkmark} \wedge \mathsf{PAY\_R}^{\text{fail}}$
$\mathsf{OCC}^{\times} \vee \mathsf{PAY\_R}^{blk}$
$s^{12}_{\perp^t}$ $\perp^t_{12}$
$s^{12}_{\perp^p}$ $\perp^p_{12}$
$s_{\top^t}$ $\top^t$
$s_{\top^p}$ $\top^p$
$s^1_{\perp^t}$ $\perp^t_1$
$s^1_{\perp^p}$ $\perp^p_1$
$s^2_{\perp^t}$ $\perp^t_2$
$s^2_{\perp^p}$ $\perp^p_2$
$*$

Figure 15: 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.

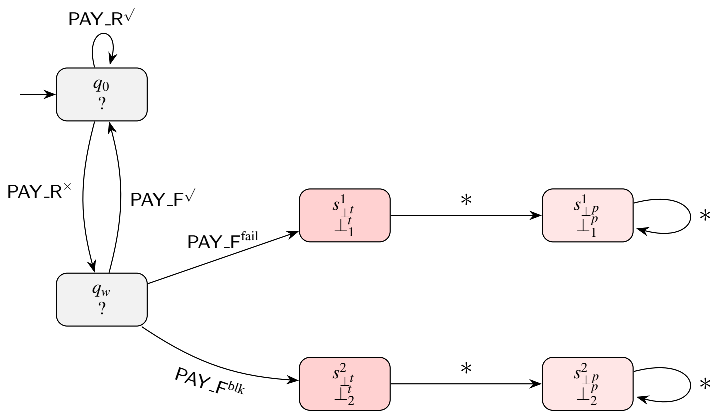

$\mathsf{PAY\_R}^{\checkmark}$
q_0
?
$\mathsf{PAY\_R}^{\times}$
$\mathsf{PAY\_F}^{\checkmark}$
$\mathsf{PAY\_F}^{\text{fail}}$
q_w
?
$\mathsf{PAY\_F}^{blk}$
$s^1_{\perp^t}$ $\perp^t_1$
$s^1_{\perp^p}$ $\perp^p_1$
$s^2_{\perp^t}$ $\perp^t_2$
$s^2_{\perp^p}$ $\perp^p_2$
$*$

Figure 16: 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 $\perp^p_1$.

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 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 The next lemma makes this coherence property explicit.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 16), 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 the aggregation of faults. 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, judgement structure than the one presented here.

9 Quantitative Violation Semantics

Motivation for Quantitative Semantics. The fundamental limitation of the forward-looking tight satisfaction semantics lies in its binary and prefix-closed nature. Specifically, the monitoring process halts definitively at the first tight violation, effectively freezing the verdict even if subsequent events in a long-running trace would constitute further breaches. While computationally efficient for runtime enforcement, this "first-fail" approach masks the full history of non-compliance. It fails to capture cumulative violations or independent failures by multiple agents over time, which is a critical requirement for comprehensive legal or normative accountability. To address this limitation, one must move beyond simple boolean or blame-set verdicts to a quantitative semantics. Such an approach measures the "degree" or "count" of violations over the entire duration of the contract. Potential methods include cumulative violation counting, weighted penalties, or violation density metrics, all of which provide a more nuanced and legally robust assessment of contract performance.

Determining the Scope of Quantification. To count violations cumulatively rather than stopping at the first breach, we must determine which part of the contract is currently "in force" at any given step. A key challenge is defining the temporal window over which violations are aggregated. We identify two primary strategies:

1. **Pre-computing Contract Length (T):** One approach is to try to calculate the

precise "duration" or "length" of the contract execution beforehand. If we knew a contract C lasts exactly T steps, we could simply sum the violations over the window $[0,T]$. However, this turns out to be complicated because contracts involving regular expressions (like guards or triggers) or unbounded repetitions ($\mathbf{Rep}(C)$) have variable lengths that depend entirely on the runtime input of the agents. The "length" is not static; it is dynamic and trace-dependent.

2. **On-the-Fly End Detection:** A more flexible alternative is to compute the end of the contract dynamically. At each step, the monitor checks if the current prefix constitutes a "complete execution" or "end" of the contract before deciding to move to the continuation. This is essentially a greedy or localized version of a "Best-Split" logic: as soon as the contract can be considered finished (even with violations), we switch context to the next phase.

In this work, we opt for **On-the-Fly End Detection** as it allows for a more responsive and adaptable monitoring strategy suitable for runtime verification of dynamic behaviors.

First, we formalize the **Contract Progress Monitor** (CPM). This monitor relies on a progression function $\mathsf{Prog} : \Gamma^* \times \mathsf{TACNL} \to \mathsf{TACNL}$ that, for any finite prefix π and a contract C, returns the active remaining parts or residual of the contract as residual contract. Unlike simple binary monitors, the CPM identifies which specific sub-contract or obligation is currently *in force* given the history of events—distinguishing, for instance, whether a sequential contract $C_1;C_2$ is currently enforcing the initial phase C_1 or has progressed to the continuation C_2. Second, we define the **Quantitative Violation Semantics** to establish how violation scores are calculated based on these residuals. Finally, we present the **Quantitative Blame Semantics** and its corresponding automata construction. By leveraging the structural tracking of the CPM to identify exactly which obligations were active at each step, this framework enables the cumulative counting of violations and the precise attribution of blame to agents over the entire duration of a contract.

Section Plan. Similarly to the methodology established in the previous section on forward-looking tight semantics, we structure our investigation of quantitative semantics as follows. We begin by defining the **Contract Progress Monitor** function, which serves as the operational foundation for tracking contract evolution. Next, we formally introduce the **Quantitative Violation Semantics** to define the counting mechanism for breaches, followed by the construction of the corresponding automata. Finally, we extend this framework to **Quantitative Blame Semantics**, detailing how cumulative scores are attributed to specific agents, and conclude with the construction of the quantitative blame monitor.

9.1 Contract Progress Monitor

The core of the quantitative framework is the ability to track the state of the contract as it evolves over time. We formalize this using a *progression function* that computes the residual contract after observing a trace prefix.

Definition 50 (Contract Progression Function)**.** *Let $\Gamma = 2^{\Sigma}$ be the event alphabet and let $\mathscr{C}$ be the set of all contracts in* TACNL *extended with a special symbol $\varepsilon_{\mathscr{C}}$. The symbol $\varepsilon_{\mathscr{C}}$ signals that all parts of the contract have been enforced before (i.e., the contract is effectively empty or fully processed). The progression function* $\mathsf{Prog} : \Gamma^* \times \mathsf{TACNL} \to \mathsf{TACNL} \cup \{\varepsilon_{\mathscr{C}}\}$ *reduces the formula on the fly following the timed progress on the trace. It takes a finite trace prefix π and a contract C and returns the residual contract. 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 recursively on the structure of C:*

Literals. *For any literal ℓ:*

$$\mathsf{Prog}(\langle A \rangle, \ell) = \varepsilon_{\mathscr{C}}$$

Whether the literal is satisfied ($\langle A \rangle \models_{\top'} \ell$) or violated ($\langle A \rangle \models_{\bot'} \ell$), it is consumed by the step A. The distinction between satisfaction and violation is handled by the quantitative scoring function, not the structural progression.

Conjunction.

$$\mathsf{Prog}(\langle A \rangle, C_1 \wedge C_2) = \mathsf{Prog}(\langle A \rangle, C_1) \wedge \mathsf{Prog}(\langle A \rangle, C_2)$$

With the identity: $\varepsilon_{\mathscr{C}} \wedge C' \equiv C'$ and $C' \wedge \varepsilon_{\mathscr{C}} \equiv C'$.

Sequence. *For a sequence $C_1;C_2$, the progression depends on whether C_1 has been fully discharged (reduced to $\varepsilon_{\mathscr{C}}$).*

$$\mathsf{Prog}(\langle A \rangle, C_1;C_2) = \begin{cases} \mathsf{Prog}(\langle A \rangle, C_1);C_2 & \textit{if } \mathsf{Prog}(\langle A \rangle, C_1) \neq \varepsilon_{\mathscr{C}}, \\ \mathsf{Prog}(\langle A \rangle, C_2) & \textit{if } \mathsf{Prog}(\langle A \rangle, C_1) = \varepsilon_{\mathscr{C}} \textit{ (immediate hand-over).} \end{cases}$$

Technically, if C_1 finishes exactly at step A, the residual is C_2. If C_1 takes multiple steps, we remain in the sequence structure.

Reparation.

$$\mathsf{Prog}(\langle A \rangle, C_1 \blacktriangleright C_2) = \begin{cases} \mathsf{Prog}(\langle A \rangle, C_1) \blacktriangleright C_2 & \textit{if } \langle A \rangle \models_{?} C_1 \textit{ (still pending),} \\ \mathsf{Prog}(\langle A \rangle, C_2) & \textit{if } \langle A \rangle \models_{\bot'} C_1 \textit{ (failure triggers repair),} \\ \varepsilon_{\mathscr{C}} & \textit{if } \langle A \rangle \models_{\top'} C_1 \textit{ (success clears all).} \end{cases}$$

Repetition.

$$\mathsf{Prog}(\langle A \rangle, C^n) = \mathsf{Prog}(\langle A \rangle, C);C^{n-1}$$
$$\mathsf{Prog}(\langle A \rangle, \mathbf{Rep}(C)) = \mathsf{Prog}(\langle A \rangle, C);\mathbf{Rep}(C)$$

Guarded Contract ($\lceil re \rceil C$). *The guard restricts C to hold only while the trace matches prefixes of re. Let $re' = \delta(re, A)$ be the Brzozowski derivative of re with respect to A.*

$$\mathsf{Prog}(\langle A \rangle, \lceil re \rceil C) = \begin{cases} \varepsilon_{\mathscr{C}} & \textit{if } \langle A \rangle \models_{\top'} \lceil re \rceil C, \\ \lceil \mathsf{Prog}_{re}(re') \rceil \mathsf{Prog}(\langle A \rangle, C) & \textit{otherwise (guard continues).} \end{cases}$$

Note: For regular expressions, $\models_{\top'} re'$ implies $\varepsilon \in \mathscr{L}(()\, re')$.

Triggered Contract ($\langle\langle re\rangle\rangle C$). *The trigger activates C once re matches. Let $re' = \delta(re, A)$.*

$$\mathsf{Prog}(\langle A\rangle, \langle\langle re\rangle\rangle C) = \begin{cases} \varepsilon_{\mathscr{C}} & \textit{if } \langle A\rangle \models_{\perp'} re \textit{ (trigger impossible)}, \\ C & \textit{if } \langle A\rangle \models_{\top'} re' \textit{ (Trigger fires)}, \\ \langle\langle \mathsf{Prog}_{re}(re)\rangle\rangle C & \textit{if } \langle A\rangle \models_{?} re. \end{cases}$$

Where $\mathsf{Prog}_{re} : 2^{\Gamma} \times \mathsf{TACNL}_{re} \to \mathsf{TACNL}_{re}$ *is defined only when* $\langle A\rangle \models_{?} \mathsf{re}$ *(where* $\langle A\rangle$ *is the trace consisting of the single event A).*

Concatenation *x For* $\mathsf{re} \cdot \mathsf{re}'$*, the progression is defined as:*

$$\mathsf{Prog}_{re}(A, \mathsf{re} \cdot \mathsf{re}') = \mathsf{Prog}_{re}(A, re) \cdot \mathsf{re}_2$$

where $\nu(\psi_1) = \varepsilon$ *if* $\varepsilon \in \mathscr{L}(\psi_1)$*, and* $\emptyset$ *otherwise.*

Union *For* $\psi = re \mid re'$*, the progression is defined by distinguishing cases based on violation:*

$$\mathsf{Prog}_{re}(A, re \mid re') = \begin{cases} \mathsf{Prog}_{re}(A, re') & \textit{if } \langle A\rangle \models_{\perp'} re \\ \mathsf{Prog}_{re}(A, re) & \textit{if } \langle A\rangle \models_{\perp'} re' \\ \mathsf{Prog}_{re}(A, re) \mid \mathsf{Prog}_{re}(A, re') & \textit{otherwise} \end{cases}$$

Kleene Star *For* re^**, the progression is defined as:*

$$\mathsf{Prog}_{re}(A, re^*) = \mathsf{Prog}_{re}(A, re) \cdot re^*$$

This definition ensures that for any trace π, $\mathsf{Prog}(\pi, C)$ returns exactly the part of the contract that remains to be enforced in the continuation π', filtering out parts that have already been processed (successfully or not).

References

[1] Rajeev Alur and David L. Dill. A theory of timed automata. In *Real Time Systems Symposium*, pages 182–193, 1990.

[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] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for distributed systems. In *Proceedings of the 6th International Conference on Software Engineering and Formal Methods*, pages 169–187, 2008.

[6] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.

[7] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2010)*, volume 6246 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2010.

[8] E. Allen Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, B:995–1072, 1990.

[9] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[10] 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.

[11] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[12] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[13] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 854–860, 2013.

[14] Guido Governatori and Zoran Milosevic. A formal analysis of a business contract language. *Information Systems Frontiers*, 8(3):273–302, 2006.

[15] Nicolas Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer Academic Publishers, 1993.

[16] David Harel and Amir Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI Series*, pages 477–498. Springer, 1985.

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

[18] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, MA, 2 edition, 2001.

[19] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[20] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[21] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[22] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[23] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yoshio Tsivinsky and Stavros Tripakis, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS 2004)*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004.

[24] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[25] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[26] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[27] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, Boston, MA, 3 edition, 2012.

[28] Ken Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

[29] Moshe Y. Vardi. Automata theory and model checking. In *Proceedings of the 8th International Conference on Computer Aided Verification*, pages 2–7, 1996.