Trace Logic and Theory of Computation

Abstract

This paper gives a self-contained account of *trace logic and computation*. Its mathematical starting point is the first-return trace of a Markov kernel on a visible subset of states. Its computational claim is that the trace operation yields a candidate foundation for computation theory.

The paper is organized around five steps. First, it defines *effective countable-state trace kernels*, i.e., machine-presentable stochastic systems with explicit input, output, halting, and visible-state structure. Second, it proves a full-abstraction theorem: the observational preorder induced by finite visible experiments is exactly trace order, equivalently exact first-return hidden-step simulation. Third, it extends the framework from closed kernels to open input/output transducers and proves that the resulting objects form a strict symmetric monoidal category, whose guarded fragment is traced. Fourth, it develops a complexity theory that splits resources into *visible* and *hidden* parts, showing both how ordinary classes are recovered and what genuinely new distinctions are preserved. Fifth, it supplies a proof system and an indexed internal logic that make precise the slogan *globally non-Boolean, locally Boolean*.

This presentation incorporates six expository improvements. It begins with a reader's guide and a dependency diagram; it carries one red–green–blue running example through all five steps; it includes a notation table and a glossary; it clearly separates formal theorems from the larger motivating program; it explicitly distinguishes three meanings of "trace" and two meanings of "logic"; and it explains the local-Boolean structure with a picture.

1 Introduction

1.1 The basic idea

A Markov chain is a system that moves from state to state with given probabilities. If we can see *all* its states, then its dynamics is just the ordinary dynamics of the chain. But often we cannot see all the states. Some are hidden. The central question of trace theory is:

> If we watch a stochastic system only through a chosen visible subset of states, what effective dynamics do we observe?

The key point is that one does *not* get the right answer by simply deleting the hidden states. A process can wander through those states before it returns to something visible. The correct construction is *first return*: start from a visible state, wait until the process returns to the visible set, and record where it returns. The resulting visible-only kernel is the *trace*.

A useful mental picture is a subway system. You see the train only at above-ground stations. The tunnels are hidden. A visible macro-step is "leave one above-ground station and next emerge at another above-ground station." The tunnels are not ignored; they are compressed into an effective visible transition law.

This idea already appears in the source paper *Traces of Consciousness*, where trace order is introduced on Markov kernels and interpreted as a logic of observation. The present paper develops only the *formal computation-theoretic* side of that idea. It does not attempt to settle questions about physics, consciousness, or empirical tests. Those larger ambitions remain motivation, not theorem.

1.2 What this paper proves and what it does not prove

The paper proves five groups of results.

(i) A class of effective trace kernels matches deterministic, nondeterministic, and probabilistic Turing-style computation at the appropriate semantic level.

(ii) The exact preorder induced by finite visible observation is trace order itself.

(iii) Open trace transducers admit sequential composition, parallel composition, and guarded feedback, and form a traced symmetric monoidal category.

(iv) Trace computation has a natural complexity theory with visible and hidden resources.

(v) Trace computation has both an internal logic of observer windows and an external proof system for refinement.

The paper *does not* prove that trace logic has already replaced the standard foundations of computation theory. It proposes and develops a mathematically coherent *formal program*: a candidate foundation in which observation is built into the model rather than added afterwards.

1.3 Three levels of discussion

The topic naturally lives at three different levels.

(1) **Motivational level.** The source paper asks foundational questions about observers, conscious agents, and physics.

(2) **Markov-kernel level.** One can study first-return traces and trace order purely as mathematics on kernels.

(3) **Computation-theoretic level.** One can ask which effectively presented trace systems count as computations, what refinement means, how systems compose, how complexity should be measured, and what proof theory they support.

This paper works primarily at level (3), using level (2) as mathematical input and level (1) only as motivation.

1.4 How to read this paper

The paper is organized to make the five-step structure explicit.

$$\text{effective model} \longrightarrow \text{full abstraction} \longrightarrow \text{composition} \longrightarrow \text{complexity} \longrightarrow \text{proof theory}$$

A reader interested mainly in the big picture can read Sections 1, 2, and 8. A reader interested mainly in the mathematics can read Sections 3–7 in order.

Section 1: sets the scope, warns about overloaded terminology, and gives a reader's guide;

Section 2: introduces one running red–green–blue example and computes its traces explicitly;

Section 3: defines effective trace kernels and states the equivalence with Turing-style computation;

Section 4: defines finite-window observers and proves full abstraction;

Section 5: extends the theory to open transducers and compositional algebra;

Section 6: develops trace complexity and shows how standard classes are recovered;

Section 7: gives a proof system and the indexed logic of observer windows;

Section 8: explains the significance and the limitations of the framework.

1.5 A warning about overloaded terms

The critique of the earlier draft correctly noted that the word *trace* was being used in three different ways, and the word *logic* in two different ways. In this revision I keep them separate.

- **First-return trace of a kernel.** For a Markov kernel P and a visible subset W, the notation
$$\mathrm{Trace}_W(P)$$
means the first-return visible kernel on W.
- **Feedback trace in a traced monoidal category.** For an open transducer $F : X \oplus U \to Y \oplus U$, the notation
$$\mathrm{Feedback}_U(F)$$
means the feedback operator that hides the interface U.
- **Matrix trace.** For an ordinary square matrix A, the notation
$$\mathrm{tr}(A)$$
means the sum of its diagonal entries.

Likewise there are two meanings of *logic*.

- **Order-theoretic logic.** The preorder $P \leq_t Q$ is read as entailment or refinement.
- **Proof-theoretic logic.** Later, formulas, derivations, soundness, completeness, and decidability are introduced as a proof system.

These meanings are related, but they are not identical.

1.6 Notation

Notation	Meaning
S	countable state set
$W \subseteq S$	visible window, or observer window
τ_W^+	first return time to W
$\mathrm{Trace}_W(P)$	first-return trace of kernel P on W
$P \leq_t Q$	P is a trace of Q
$P \sqsubseteq_{\mathsf{ctx}} Q$	observational contextual preorder
δ	sub-Markov kernel of an open transducer
$\mathrm{Feedback}_U$	guarded feedback hiding interface U
T_{raw} and T_W	raw and visible halting times
D_W	hidden-state depth relative to W
$\mathbb{B}(M)$	Boolean algebra of effective windows below M

1.7 Glossary of ten core terms

Term	Informal meaning
visible state	a state the observer is allowed to see
hidden state	a state used internally between visible events
first return	the next time the process re-enters the visible set
trace	the effective visible dynamics induced by first return
trace order	the refinement relation "is a trace of"
contextual preorder	agreement on all finite visible experiments
effective kernel	a kernel whose states, successors, and probabilities are computable
transducer	an open computation with inputs, outputs, internal state, and halting
visible cost	resource cost measured at the visible level
local Booleanity	Boolean algebra of windows below one fixed ambient computation

2 A running example: red, green, and blue

This section introduces one toy example that will be used throughout the rest of the paper.

Example 2.1 (A three-color kernel)**.** *Let*

$$Q = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{3}{4} & 0 \\ \frac{1}{3} & \frac{2}{3} & 0 \end{pmatrix}$$

on the state space

$$S = \{r, g, b\},$$

where r, g, and b stand for red, green, and blue.

If we watch the whole chain, then Q is the dynamics. But if we attend only to two of the three colors, then the third color becomes hidden and we must compute a first-return trace.

2.1 First-return traces of the color chain

For a visible subset $W \subseteq S$, define

$$\tau_W^+ := \inf\{n \geq 1 : X_n \in W\}, \qquad \mathrm{Trace}_W(Q)(u,v) := \mathbb{P}_u[X_{\tau_W^+} = v].$$

For finite chains the trace can be written in block form as

$$\mathrm{Trace}_W(Q) = a + b(I-c)^{-1}d,$$

where a records direct visible-to-visible transitions, b visible-to-hidden transitions, c hidden-to-hidden transitions, and d hidden-to-visible transitions.

For the windows

$$W_{rg} = \{r, g\}, \qquad W_{rb} = \{r, b\}, \qquad W_{gb} = \{g, b\},$$

a direct calculation gives

$$Q_{rg} := \mathrm{Trace}_{W_{rg}}(Q) = \begin{pmatrix} \frac{7}{12} & \frac{5}{12} \\ \frac{1}{4} & \frac{3}{4} \end{pmatrix},$$

$$Q_{rb} := \mathrm{Trace}_{W_{rb}}(Q) = \begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ 1 & 0 \end{pmatrix}, \qquad Q_{gb} := \mathrm{Trace}_{W_{gb}}(Q) = \begin{pmatrix} \frac{7}{8} & \frac{1}{8} \\ \frac{5}{6} & \frac{1}{6} \end{pmatrix}.$$

These three traces are distinct. So one ambient process already yields several different observer-relative computations.

2.2 What the example teaches

The example already shows three important ideas.

(1) **Hiding changes the probabilities.** Watching only red and green does not merely delete blue; it changes the red/green transition law.

(2) **One ambient machine gives many visible machines.** The windows W_{rg}, W_{rb}, and W_{gb} produce different visible kernels.

(3) **Observation is window-relative.** What the system "looks like" depends on what is visible.

If we care about finite temporal observations, then the visible word

$$(r, g, r)$$

in the red–green window has probability

$$Q_{rg}(r,g)\, Q_{rg}(g,r) = \frac{5}{12} \cdot \frac{1}{4} = \frac{5}{48}.$$

This is exactly the kind of finite visible experiment used later in the full-abstraction theorem.

2.3 Why this example will reappear

The same red–green–blue system will be used

- as a finite rational effective trace kernel in Section 3,
- as a visible-observer example in Section 4,
- as an open transducer in Section 5,
- as a visible-versus-hidden cost example in Section 6,
- and as a window algebra inside an ambient computation in Section 7.

3 Effective trace kernels as computations

3.1 Probability presentation classes

To compare trace systems with ordinary computation, one must first restrict the transition probabilities to those that are effectively presented. Fix a class

$$\mathsf{P} \in \{\mathrm{Rat}, \mathrm{Alg}, \mathrm{Comp}, \mathrm{Comp}^O\}.$$

The intended readings are:

- Rat: exact rational probabilities;
- Alg: real algebraic probabilities;
- Comp: computable real probabilities;
- Comp^O: computable real probabilities relative to a fixed oracle O.

This effectivity restriction matters. If one allowed arbitrary real probabilities, an infinite-state Markov kernel could encode noncomputable information. So the correct claim is *not* that all countable-state Markov kernels are Turing machines. The correct claim is that *machine-presentable* trace kernels correspond to Turing-style computation at matching randomness or oracle strength.

3.2 Effective countable-state computation kernels

Definition 3.1 (P-effective countable-state computation kernel)**.** *Fix input alphabet Σ and output alphabet Γ. A* P-effective countable-state computation kernel *is a tuple*

$$K = (S, m, \sigma, p, H, \iota, \mathrm{out})$$

such that:

(1) $S \subseteq \mathbb{N}$ is a decidable countable state set;

(2) each $s \in S$ has a finite computable successor list

$$\sigma(s, 1), \ldots, \sigma(s, m(s)) \in S$$

with computable branching degree $m(s) \geq 1$;

(3) each row probability $p_{s,i}$ is given uniformly as a P*-name and satisfies*

$$\sum_{i=1}^{m(s)} p_{s,i} = 1;$$

(4) $H \subseteq S$ is a decidable halting set and every $h \in H$ is absorbing;

(5) $\iota : \Sigma^* \to S$ is a computable input encoder;

(6) $\mathrm{out} : H \to \Gamma^* \cup \{\mathsf{acc}, \mathsf{rej}\}$ is a computable output decoder.

This data induces a Markov process

$$X_0 = \iota(x), \qquad X_{n+1} \sim P(X_n, \cdot)$$

on each input x.

3.3 Effective trace kernels

Definition 3.2 (P-effective trace kernel)**.** *A* P-effective countable-state trace kernel *is a pair (K, V) where K is a* P*-effective computation kernel and $V \subseteq S$ is a decidable visible set satisfying*

$$\iota(\Sigma^*) \subseteq V, \qquad H \subseteq V,$$

such that:

(1) for every $v \in V$, the first return time

$$\tau_V^+ := \inf\{n \geq 1 : X_n \in V\}$$

is almost surely finite;

(2) the first-return visible kernel

$$\mathrm{Trace}_V(P)(v, v') := \mathbb{P}_v[X_{\tau_V^+} = v']$$

is again uniformly P*-effective.*

If $V = S$, nothing is hidden and one gets ordinary step-by-step semantics. If $V \subsetneq S$, the hidden states act as internal microstates and the trace gives a macro-step semantics.

Remark 3.3. *Finite branching is the right machine-like local model. One can enlarge the traced side to countably supported rows, but finite branching is the clean base class for direct comparison with ordinary Turing machines.*

3.4 From Turing machines to effective kernels

Deterministic, nondeterministic, and probabilistic Turing machines all embed naturally into this framework.

Theorem 3.4 (Effective equivalence: machine to kernel direction)**.** *Fix a presentation class* P*.*

(i) Every deterministic Turing machine is exactly a point-mass effective kernel.

(ii) Every finitely branching nondeterministic Turing machine is exactly a positive-support effective kernel under support semantics.

(iii) Every P*-probabilistic Turing machine is exactly a* P*-effective stochastic kernel under full path-law semantics.*

In each case, the natural encoding preserves accepted language, halting behavior, output semantics, and runtime up to the stated encoding overhead.

Proof sketch. For a deterministic machine, take the configuration graph as the state set and put probability 1 on the unique next configuration. Halting configurations become absorbing.

For a nondeterministic machine, assign a positive weight to each legal next branch; uniform weights are the simplest. What is preserved canonically is support: an output is reachable in the nondeterministic sense exactly when it occurs with positive probability in the kernel.

For a probabilistic machine, again use configurations as states, but let each row of the kernel be the machine's actual one-step branching law. Then finite path probabilities agree exactly. Hence output distribution, halting-time distribution, and acceptance probabilities agree as well. □

3.5 From effective kernels back to machines

Theorem 3.5 (Effective equivalence: kernel to machine direction)**.** *Every point-mass effective kernel is simulated by a deterministic Turing machine. Every finitely supported support kernel is simulated by a nondeterministic Turing machine. Every* P*-effective finitely supported stochastic kernel is simulated by a* P*-probabilistic Turing machine using an exact fair-bit sampler.*

Proof sketch. The deterministic and nondeterministic cases are immediate from successor enumeration.

In the stochastic case, suppose a row distribution has cumulative probabilities

$$0 = c_0 < c_1 < \cdots < c_m = 1.$$

Generate a random binary real

$$U = 0.b_1 b_2 b_3 \cdots$$

from fair bits. Approximate the boundaries c_i until the current dyadic interval for U lies strictly inside one interval $[c_{i-1}, c_i)$. This happens almost surely and returns index i with probability exactly p_i. Therefore the simulated machine has exactly the same path law as the kernel. □

Corollary 3.6 (What is computationally equivalent to what)**.** *The correct equivalence statement is*

effective trace kernels = Turing-style computations at matching semantic strength,

not

all countable-state Markov kernels = all Turing machines.

3.6 Trace in Turing-machine language

If M is a Turing machine and $V \subseteq \mathrm{Conf}(M)$ is a decidable set of checkpoint configurations containing the start and halting configurations, define

$$\mathrm{Trace}_V(M)(c, c') := \mathbb{P}_M[C_{\tau_V^+} = c' \mid C_0 = c].$$

Then trace means:

hide internal microsteps and retain only the checkpoint-to-checkpoint first-return dynamics.

That is the exact computational reading of first-return trace.

3.7 The running example as an effective kernel

The red–green–blue kernel Q from Section 2 is a finite rational trace kernel. So it already lies in the simplest effective class. It has no input or halting structure because it is being used as a closed stochastic computation for intuition, but it can be embedded into the full model simply by taking a singleton input alphabet, a trivial encoder, and no halting states.

4 Observation, refinement, and full abstraction

4.1 Visible semantics

For refinement questions, only the visible behavior matters.

Definition 4.1 (Effective visible kernel)**.** *An* effective visible kernel *is a triple*

$$P = (S, K, I)$$

where $S \subseteq \mathbb{N}$ is decidable, K is an effective Markov kernel on S, and $I \subseteq S$ is a decidable visible interface such that every observer window $W \subseteq I$ under consideration has an effective first-return trace $\mathrm{Trace}_W(K)$*.*

Definition 4.2 (Trace order relative to an embedding)**.** *Let P and Q be effective visible kernels and let*

$$j : I_P \hookrightarrow I_Q$$

be an interface-preserving injection. Then

$$P \leq_t^j Q$$

means

$$P = j^*\Big(\mathrm{Trace}_{j(I_P)}(Q)\Big)\,,$$

i.e. P is exactly the visible first-return behavior of Q through the embedded interface.

4.2 Finite-window observers

Definition 4.3 (Basic finite-window observer)**.** *A basic observer context over P is a pair*

$$C = (W, w)$$

where $W \subseteq I_P$ is decidable and

$$w = (w_0, \ldots, w_n) \in W^{n+1}$$

is a finite visible word. Its observed value is

$$\mathrm{Obs}_C(P) := \mathbb{P}_{w_0}[(Y_0, \ldots, Y_n) = w]\,,$$

where (Y_k) is the Markov chain with kernel $\mathrm{Trace}_W(P)$*.*

By the Markov property,

$$\mathrm{Obs}_{(W,(w_0,\dots,w_n))}(P) = \prod_{i=0}^{n-1} \mathrm{Trace}_W(P)(w_i, w_{i+1}).$$

Definition 4.4 (Observer-induced contextual preorder)**.** *Let P, Q be effective visible kernels and let $j : I_P \hookrightarrow I_Q$. Define*

$$P \sqsubseteq^{j}_{\mathsf{ctx}} Q$$

iff for every basic finite-window observer context C over P,

$$\mathrm{Obs}_C(P) = \mathrm{Obs}_{j(C)}(Q).$$

So $P \sqsubseteq^{j}_{\mathsf{ctx}} Q$ says that every finite visible experiment available on P is reproduced exactly by Q through the embedding j.

4.3 Full abstraction

Lemma 4.5 (One-step basis)**.** *If for every $u, v \in I_P$,*

$$\mathrm{Obs}_{(I_P,(u,v))}(P) = \mathrm{Obs}_{(j(I_P),(j(u),j(v)))}(Q),$$

then

$$P \leq^{j}_{t} Q.$$

Proof. The left-hand side is exactly $P(u, v)$, and the right-hand side is exactly the $(j(u), j(v))$-entry of $\mathrm{Trace}_{j(I_P)}(Q)$. Equality of all one-step observations therefore gives equality of the whole visible kernel. □

Theorem 4.6 (Full abstraction)**.** *For effective visible kernels P, Q and embedding $j : I_P \hookrightarrow I_Q$,*

$$P \leq^{j}_{t} Q \iff P \sqsubseteq^{j}_{\mathsf{ctx}} Q.$$

Proof. If $P \leq^{j}_{t} Q$, then tracing Q on smaller windows commutes with pullback along j, because trace-of-trace is trace. Therefore every finite visible cylinder probability agrees.

Conversely, if every finite visible observation agrees, then in particular every one-step visible observation agrees. By the one-step basis lemma, $P \leq^{j}_{t} Q$. □

Corollary 4.7 (What the observational preorder really is)**.** *The exact preorder induced by finite-window observation is trace order itself. Equivalently, it is* exact first-return hidden-step simulation*.*

Remark 4.8 (The empirical caveat behind the one-step basis)**.** *Mathematically, one visible macro-step determines the traced kernel. Empirically, however, estimating even one macro-step probability may require very long raw observation, because hidden excursions can be arbitrarily long. So "one visible step is enough" is a theorem about semantics, not about laboratory convenience.*

4.4 What kind of refinement is trace order?

The full-abstraction theorem lets us classify trace order precisely.

Proposition 4.9. *Trace order is:*

(i) exactly exact first-return hidden-step simulation;

(ii) not weak bisimulation, because it is asymmetric and has strict pairs;

(iii) strictly stronger than qualitative stuttering refinement, because it preserves exact probabilities rather than merely positive support;

(iv) locally an exact abstract interpretation, but not globally reducible to one fixed abstraction domain.

Proof sketch. Clause (i) is the theorem just proved. Clause (ii) follows because weak bisimulation is an equivalence relation, while trace order has strict comparable pairs. Clause (iii) follows from examples where two systems allow the same visible words but with different probabilities. Clause (iv) is made precise by the local subset representation below. □

Proposition 4.10 (Local subset representation)**.** *Fix one ambient computation Q. For effective windows $W_1, W_2 \subseteq I_Q$,*

$$W_1 \subseteq W_2 \iff \mathrm{Trace}_{W_1}(Q) \leq_t \mathrm{Trace}_{W_2}(Q).$$

Proof. If $W_1 \subseteq W_2$, then tracing on W_2 and then on W_1 is the same as tracing directly on W_1.

Conversely, if $\mathrm{Trace}_{W_1}(Q) \leq_t \mathrm{Trace}_{W_2}(Q)$, then the support of the left-hand trace must sit inside the support of the right-hand trace, hence $W_1 \subseteq W_2$. □

4.5 The running example as a refinement example

For the color kernel Q,

$$Q_{rg} \leq_t Q, \qquad Q_{rb} \leq_t Q, \qquad Q_{gb} \leq_t Q.$$

These traces are different, so finite visible observers distinguish them. For example the one-step probability of returning from red to green is

$$Q_{rg}(r, g) = \frac{5}{12}, \qquad Q_{rb}(r, b) = \frac{1}{4},$$

so the two windows produce genuinely different visible computations.

This is the first place where the running example shows something important:

> one ambient system can have many different observer-relative visible semantics.

5 Trace transducers and compositionality

5.1 Why open systems matter

Closed kernels are not enough for a serious theory of computation. Computations have inputs, produce outputs, halt, and get connected together. So the trace framework must be extended from autonomous kernels to open channels.

5.2 Sub-Markov kernels and halting

Definition 5.1 (Effective finitely supported sub-Markov kernel)**.** *Let $A, B \subseteq \mathbb{N}$ be decidable. A* finitely supported effective sub-Markov kernel

$$K : A \rightsquigarrow B$$

assigns to each $a \in A$ a finite computable support in B together with row probabilities that sum to at most 1*. The missing mass*

$$\lambda_K(a) := 1 - \sum_{b \in B} K(a, b)$$

is its halting deficiency.

Sub-Markov rather than Markov kernels are the right open-system object because missing mass propagates halting compositionally.

5.3 Macro and micro forms

Definition 5.2 (Macro trace transducer)**.** *A* P-effective trace transducer *of type $X \to Y$ is a triple*

$$T = (S, s_0, \delta)$$

where $S \subseteq \mathbb{N}$ is a decidable live-state set, $s_0 \in S$ is an initial state, and

$$\delta : S \times X \rightsquigarrow Y \times S$$

is a finitely supported effective sub-Markov kernel.

Intuitively, on input x and current state s, the kernel δ chooses an output y and a next live state s', or halts with the missing probability.

Definition 5.3 (Micro implementation)**.** *A* micro implementation *of type $X \to Y$ is a square hidden-state Markov kernel with visible input configurations, visible output configurations, hidden microstates, and halting states, such that the first visible return from an input configuration is always an output configuration or a halting state.*

Proposition 5.4 (Micro and macro forms are equivalent)**.** *Every micro implementation induces a macro trace transducer by first-return tracing. Conversely, every macro trace transducer admits a trivial one-step micro realization.*

Proof. The forward direction is simply the definition of macro behavior as first return from the visible input interface to the visible output interface.

For the reverse direction, realize each macro-step in one physical step by adding a fresh halting state and making the output configurations absorbing. □

5.4 Finite-word semantics

Let

$$T = (S, s_0, \delta) : X \to Y.$$

On an input word $u = x_1 \cdots x_n$, define subdistributions μ_k on $Y^k \times S$ by

$$\mu_0(\varepsilon, s) = \begin{cases} 1, & s = s_0, \\ 0, & s \neq s_0, \end{cases}$$

and

$$\mu_{k+1}(wy, s') = \sum_{s \in S} \mu_k(w, s)\, \delta((s, x_{k+1}), (y, s')).$$

Then:

- $\mu_n(w, s)$ is the probability of still being live in state s after output word w;
- $\|\mu_n\|$ is the probability of not yet halting by step n;
- $1 - \|\mu_n\|$ is the probability of halting by or before step n.

So inputs, outputs, and halting are all handled by the same sub-Markov semantics.

5.5 Sequential composition

Definition 5.5 (Sequential composition)**.** *If*

$$T = (S, s_0, \delta_T) : X \to Y, \qquad U = (R, r_0, \delta_U) : Y \to Z,$$

their composite

$$U \circ T : X \to Z$$

has state space $S \times R$, initial state (s_0, r_0), and kernel

$$\delta_{U \circ T}((s, r), x)(z, (s', r')) = \sum_{y \in Y} \delta_T((s, x), (y, s'))\, \delta_U((r, y), (z, r')).$$

This is just categorical composition of stateful sub-Markov kernels.

5.6 Parallel composition

The tensor on interfaces is disjoint union:

$$X \otimes X' := X \oplus X'.$$

On transducers it is asynchronous parallel composition: a left input runs only the left component, and a right input runs only the right component.

5.7 Guarded feedback

Suppose

$$F : X \oplus U \to Y \oplus U$$

is a guarded transducer. Split its macro kernel into blocks

$$a : X \to Y, \qquad b : X \to U, \qquad c : U \to U, \qquad d : U \to Y.$$

Then the feedback that hides U is

$$\mathrm{Feedback}_U(F) := a + b(I - c)^{-1} d = a + b\Big(\sum_{n \geq 0} c^n\Big) d.$$

This is exactly the first-return formula for hiding the interface U.

Theorem 5.6 (Compositional algebra of trace computation)**.** *Effective trace transducers form a category* $\mathsf{ETT}_{\mathsf{P}}$ *with sequential composition. With disjoint-sum tensor they form a strict symmetric monoidal category. Its guarded fragment* $\mathsf{GETT}_{\mathsf{P}}$*, equipped with* $\mathrm{Feedback}_U$*, satisfies the traced monoidal equations: vanishing, superposing, yanking, and trace-of-trace.*

Proof sketch. Associativity of sequential composition is associativity of finite summation. The tensor is a bifunctor because left and right sides run independently on their own tagged interfaces. The feedback equations reduce to the algebra of the series

$$(I - c)^{-1} = \sum_{n \geq 0} c^n,$$

which is precisely the same algebra that underlies first-return traces of hidden states. □

5.8 The running example as a transducer

From the red–green window of the color chain we obtain a transducer

$$T_{rg} : 1 \to \{r, g\}$$

as follows. The live state set is $\{r, g\}$, the input alphabet is the singleton $1 = \{*\}$, and

$$\delta_{rg}((s, *), (y, y)) := Q_{rg}(s, y).$$

Each input pulse triggers one visible macro-step of the red–green trace and outputs the next visible color. Thus the same running example that began as a closed trace kernel becomes an open computation.

6 Complexity of trace computation

6.1 Visible and hidden resources

In ordinary complexity theory, one usually counts time and space from the point of view of the entire machine. Trace computation splits those resources into two layers:

(i) **visible resources**, determined by the first-return computation seen by the observer;

(ii) **hidden resources**, spent in the microdynamics between visible returns.

That split is the conceptual heart of trace complexity.

6.2 Basic definitions

Fix an effective trace machine with raw process (X_n^x) and visible window W.

Definition 6.1 (Raw and visible time)**.** *The raw halting time on input* x *is*

$$T_{\mathrm{raw}}(x) := \inf\{n \geq 0 : X_n^x \in H\}.$$

The visible halting time is

$$T_W(x) := \inf\{k \geq 0 : Y_k^W \in H\},$$

where Y_k^W *is the* k*th visible return to* W*.*

Definition 6.2 (Hidden depth)**.** *The depth of the kth visible block is*

$$\Delta_k^W(x) := \tau_{k+1}^W - \tau_k^W,$$

and the hidden-state depth of the run is

$$D_W(x) := \max_{0 \le k < T_W(x)} \Delta_k^W(x).$$

Definition 6.3 (Visible and raw space)**.** *Fix a size norm $\|-\|$ on states. Then*

$$S_{\mathrm{raw}}(x) := \max_{0 \le n < T_{\mathrm{raw}}(x)} \|X_n^x\|, \qquad S_W(x) := \max_{0 \le k < T_W(x)} \|Y_k^W\|.$$

One can likewise define visible randomness cost, finite-window sample complexity, and visible entropy rate.

6.3 What is an invariant of trace semantics?

Theorem 6.4 (Visible costs are extensional invariants)**.** *Any complexity quantity that depends only on the visible first-return kernel is invariant under trace equivalence. In particular, visible output law, time-to-visible-output, visible randomness cost, finite-window sample complexity, and visible entropy rate are extensional invariants of trace semantics.*

Proof. If two implementations have the same visible first-return kernel, then they induce the same visible chain, the same visible cylinder probabilities, and the same stationary visible process. Every quantity listed in the theorem is computed from that visible data alone. □

6.4 Local monotonicities

Suppose $W_1 \subseteq W_2$ are two observer windows inside one ambient machine.

Proposition 6.5 (Local trace-order monotonicities)**.** *For every run,*

$$T_{W_1} \le T_{W_2}, \qquad S_{W_1} \le S_{W_2}, \qquad D_{W_1} \ge D_{W_2}.$$

Moreover,

$$T_{\mathrm{raw}} = \sum_{k < T_W} \Delta_k^W \le T_W \, D_W.$$

Proof. Shrinking the visible window deletes some visible returns, so fewer visible macro-steps are counted and the visible state set can only shrink. But each surviving visible block is now a concatenation of one or more old blocks, so hidden depth can only increase. Summing block lengths gives the raw-time factorization. □

This yields a trace-theoretic uncertainty principle:

> coarser observation shortens visible time but deepens hidden implementation.

Standard complexity keeps only the product-like bound on raw time. Trace complexity keeps the pair (T_W, D_W).

6.5 What is not invariant? Hidden padding

Theorem 6.6 (Arbitrary hidden padding)**.** *Fix a visible window W and an effective trace machine M. For any integers $m, N \geq 1$ there exists a machine $M_{m,N}$ such that:*

(1) $\mathrm{Trace}_W(M_{m,N}) = \mathrm{Trace}_W(M)$*;*

(2) every visible block of $M_{m,N}$ has length at least m;

(3) every visible block of $M_{m,N}$ visits a hidden state of size at least N.

Therefore raw time, raw space, and hidden depth are not extensional invariants.

Proof. Insert a deterministic hidden chain

$$u \to h_1 \to h_2 \to \cdots \to h_{m-1} \to z$$

before each old visible return mechanism out of a visible state u. Because the inserted path is deterministic and has no new visible exits, it does not change the first-return distribution to W. Padding the codes of the hidden states forces arbitrarily large raw space. □

6.6 A small but important correction: entropy is not monotone

An earlier conjecture said that if $P \leq_t Q$, then the entropy rate of P should be at most the entropy rate of Q. That is false.

Example 6.7 (Entropy-rate counterexample)**.** *Let*

$$Q = \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix}$$

on states $\{a, h, b\}$, and trace on the visible subset $W = \{a, b\}$. Then

$$P := \mathrm{Trace}_W(Q) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 1 & 0 \end{pmatrix}.$$

So $P \leq_t Q$, but the stationary entropy rates satisfy

$$h(P) = \frac{2}{3} > \frac{2}{5} = h(Q).$$

So entropy rate per native step is not monotone under trace order. The same example also shows that fair-bit randomness cost per native step need not be monotone.

6.7 Recovering standard complexity classes

Because effective trace kernels are equivalent to Turing-style computations, the standard classes must be recovered when the observer sees every step.

Definition 6.8 (Parameterized trace classes)**.** *Let $f, g, s : \mathbb{N} \to \mathbb{N}$.*

- $\mathsf{DTrace}(f, g, s)$ *consists of point-mass effective trace machines with visible time $O(f)$, hidden depth $O(g)$, and raw space $O(s)$.*

- $\mathsf{NTrace}(f, g, s)$ *is the analogous support-semantics class.*
- $\mathsf{BTrace}(f, g, s)$ *is the analogous bounded-error probabilistic class.*

Theorem 6.9 (Classical classes are recovered)**.** *When the observer sees every configuration, i.e. when $W = S$ and hidden depth is constant, the polynomially bounded fragments recover the usual classes:*

$$\mathsf{DTrace}(\mathrm{poly}, 1, \mathrm{poly}) = P,$$
$$\mathsf{NTrace}(\mathrm{poly}, 1, \mathrm{poly}) = NP,$$
$$\mathsf{BTrace}(\mathrm{poly}, 1, \mathrm{poly}) = BPP,$$

and the raw-space-bounded deterministic fragment recovers $PSPACE$.

Proof sketch. If $W = S$, then visible and raw steps coincide and hidden depth is 1. The effective-equivalence theorem translates such trace machines into ordinary deterministic, nondeterministic, or probabilistic Turing machines and back with only the standard polynomial overhead. □

6.8 The running example as a complexity example

Take the red–green window W_{rg} of the color chain. Now replace the hidden blue state by a deterministic chain

$$b_1 \to b_2 \to \cdots \to b_m$$

before returning to red or green with the same eventual distribution as before. The visible red–green trace is unchanged, but raw time and hidden depth increase with m.

So even this toy example shows the main lesson of trace complexity:

> two implementations can have the same visible computation and radically different hidden cost.

7 Proof theory and internal logic

7.1 Why "logic" has two layers

The order-theoretic layer is the external refinement order:

$$P \leq_t Q.$$

The proof-theoretic layer adds formulas, derivations, soundness, completeness, and decidability. Both are needed. The order says what it means for one computation to be a visible view of another; the proof theory says how to reason about that relation.

7.2 Program syntax

Definition 7.1 (Typed trace programs)**.** *Fix a typed atomic signature with symbols*

$$a : X \to Y.$$

Program terms are generated by

$$M, N ::= a \mid \mathrm{id}_X \mid N \circ M \mid M \otimes N \mid \mathrm{Feedback}_U(M),$$

with the obvious typing rules.

Interpret each atomic symbol as an effective trace transducer. Then composition, tensor, and feedback are interpreted by the operations of Section 5.

Proposition 7.2 (Structural soundness)**.** *Every equation derivable from the usual axioms of a traced symmetric monoidal category is sound for trace-program semantics.*

Proof. This is immediate because guarded effective trace transducers form a traced symmetric monoidal category. □

7.3 The internal logic of observer windows

Fix one ambient computation M with visible interface I_M.

Definition 7.3 (Admissible window algebra)**.** *Let $\mathbb{B}(M)$ be the Boolean algebra of effective visible windows*

$$W \subseteq I_M$$

for which the first-return trace $\mathrm{Trace}_W(M)$ is effective.

The key point is that *logic is local.* One does not get one giant Boolean algebra of all computations. One gets, for each fixed ambient computation M, a Boolean algebra of effective observer windows inside M.

Definition 7.4 (Local formulas)**.** *Over a fixed M, formulas are generated by*

$$A, B ::= p \mid \top \mid \bot \mid \neg A \mid A \wedge B \mid A \vee B,$$

where each atom p denotes a visible subset of I_M.

Theorem 7.5 (Local subset representation)**.** *For windows $W_1, W_2 \in \mathbb{B}(M)$,*

$$W_1 \subseteq W_2 \iff \mathrm{Trace}_{W_1}(M) \leq_t \mathrm{Trace}_{W_2}(M).$$

Hence the principal ideal

$$\downarrow M := \{P : P \leq_t M\}$$

is order-isomorphic to the Boolean algebra $\mathbb{B}(M)$.

Proof. This is exactly the local subset representation from Section 4, now read as a theorem about the logic below one ambient computation. □

Corollary 7.6 (Local Boolean soundness and completeness)**.** *Ordinary propositional calculus is sound and complete for the local entailment relation inside $\mathbb{B}(M)$.*

So the slogan

> *globally non-Boolean, locally Boolean*

means the following precise thing:

> the logic of trace computation is an indexed Boolean logic over a non-Boolean base of computations.

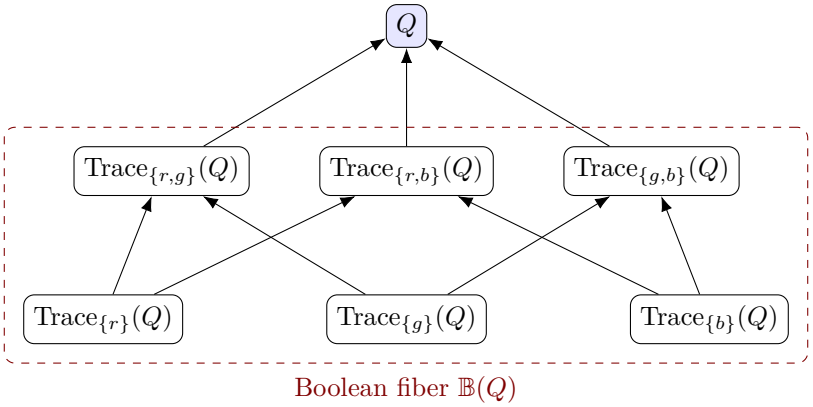


Figure 1: One ambient computation Q sits above many observer windows. Boolean operations make sense *inside* the fiber below Q, not globally across all computations at once.

7.4 A picture of local Booleanity

7.5 Quantitative observer predicates

The source paper emphasized finite temporal sampling. In the proof-theoretic setting this yields quantitative predicates.

Definition 7.7 (Finite-window observation predicates)**.** *Fix a closed computation M, a window $W \in \mathbb{B}(M)$, and a finite visible word*

$$w = (w_0, \ldots, w_n) \in W^{n+1}.$$

Define the exact cylinder probability

$$\mathrm{Obs}_W^M(w) := \mathbb{P}_{w_0}[(Y_0, \ldots, Y_n) = w]\,.$$

Atomic quantitative assertions have the form

$$\mathrm{Obs}_W^M(w) \;\bowtie\; q$$

with $\bowtie \in \{=, \leq, <, \geq, >\}$ and $q \in \mathbb{Q}$.

7.6 External refinement calculus

Definition 7.8 (External judgments)**.** *The external proof theory has judgments*

$$P \preceq^j Q \qquad \text{and} \qquad P \equiv Q,$$

where $P \preceq^j Q$ means $[\![P]\!] \leq_t^j [\![Q]\!]$ and $P \equiv Q$ means mutual refinement.

The rules come in four groups:

(1) traced symmetric monoidal equations;

(2) reflexivity and transitivity of refinement;

(3) contextuality under composition, tensor, and guarded feedback;

(4) observational extensionality: equality of all one-step visible observations implies refinement.

Theorem 7.9 (Soundness and completeness)**.** *The external refinement calculus is sound and complete:*

$$P \preceq^{j} Q \quad \textit{is derivable} \quad \Longleftrightarrow \quad [\![P]\!] \leq_t^{j} [\![Q]\!].$$

Proof sketch. Soundness is immediate from the traced-category semantics and from the meaning of the observational extensionality rule.

For completeness, if $[\![P]\!] \leq_t^{j} [\![Q]\!]$, then by full abstraction all finite-window observations coincide. In particular, all one-step observations coincide, and the observational extensionality rule derives $P \preceq^{j} Q$. □

7.7 Decidable and undecidable fragments

Theorem 7.10 (Finite rational fragment)**.** *For finite-state rational trace computations, the following problems are decidable: local entailment, quantitative model checking, refinement checking, and equivalence checking.*

Proof. In the finite rational fragment one can compute every first-return kernel exactly by rational linear algebra. Model checking reduces to finite arithmetic, and refinement checking reduces to finitely many equality checks of rational entries. □

Theorem 7.11 (Full effective fragment)**.** *For the full class of effective countable-state trace computations, model checking and complete proof search are undecidable.*

Proof. Deterministic Turing machines embed as point-mass effective trace kernels. Hence the ordinary acceptance problem reduces immediately to model checking of suitable halting predicates. □

7.8 A separation theorem

The proof theory distinguishes phenomena that are not captured by several nearby frameworks.

Theorem 7.12 (Separation theorem)**.** *The trace logic developed here expresses all of the following at once:*

(i) observer-indexed subcomputations of one and the same ambient computation;

(ii) exact first-return quantitative refinement;

(iii) local Boolean meet, join, and complement inside one ambient computation;

(iv) an external refinement preorder in addition to equational traced structure.

That combination is not available under the standard frozen-observation semantics of probabilistic automata, HMMs, or labelled MDPs; not under weak bisimulation or coalgebraic bisimulation; and not from traced-monoidal equations alone.

Proof sketch. Our running color kernel Q already gives three distinct observer windows

$$\mathrm{Trace}_{\{r,g\}}(Q), \qquad \mathrm{Trace}_{\{r,b\}}(Q), \qquad \mathrm{Trace}_{\{g,b\}}(Q)$$

below one ambient computation. A standard HMM or labelled MDP with one fixed observation map yields only one visible process at a time. Weak bisimulation is an equivalence relation and therefore cannot represent strict trace entailment. Purely qualitative support semantics cannot distinguish systems with the same visible words but different visible probabilities. Finally, traced-monoidal equations govern only equality of composites; they do not by themselves supply the local Boolean fiber below one ambient computation. □

Remark 7.13. *This comparison must be worded carefully. It does* not *say that probabilistic automata, HMMs, or MDPs cannot encode particular instances of trace computations. They often can. What it says is narrower and more important: under their* standard semantics*, they do not supply the same internal logic. If one enriches them with observer windows as first-class logical data, one is adding precisely the distinctive feature of trace logic.*

7.9 The running example as a proof-theoretic example

The color kernel Q from Section 2 gives a concrete window algebra. Inside $\mathbb{B}(Q)$ we have, for example,

$$\{r, g\} \wedge \{r, b\} = \{r\}, \qquad \{r, g\} \vee \{r, b\} = \{r, g, b\}.$$

So local Booleanity is not an abstract slogan. It is visible in the toy example itself.

8 Scope, significance, and limitations

8.1 What is genuinely new here?

Even setting aside the larger consciousness motivation, the framework contributes five conceptually new pieces.

(1) **Observation is built into the model.** Visible windows are part of the formalism, not an afterthought.

(2) **Refinement is quantitative.** The theory preserves exact first-return probabilities, not only support.

(3) **The framework is compositional.** Open trace transducers admit a clear algebra of composition.

(4) **Complexity is factored.** Visible latency and hidden implementation cost are separated rather than collapsed.

(5) **The proof theory matches the semantics.** There is both an internal window logic and an external sound/complete refinement calculus.

8.2 Why the framework is plausible

The framework is plausible for three reasons.

(i) It recovers classical computation theory when the observer sees everything.

(ii) It adds information that classical theory intentionally forgets: observer-relative latency, sampling difficulty, and hidden implementation depth.

(iii) It does not confuse philosophy with theorem. The computation-theoretic results stand on their own even if one brackets the broader metaphysical program.

8.3 What remains open

The following remain open and should not be confused with the theorems proved here.

- developing the same theory at full measurable-space generality rather than mostly in the effective countable-state regime;
- exploring synchronous as well as asynchronous monoidal structures for open systems;
- strengthening the proof theory beyond the finite rational fragment;
- relating the computational framework more directly to the empirical or physical questions raised in the source paper.

So the correct conclusion is not "the foundations have been replaced." The correct conclusion is:

> trace logic and computation is now a coherent and mathematically developed candidate foundation, with a clear model of computation, a refinement semantics, a compositional algebra, a resource theory, and a proof system.

9 Conclusion

This revised paper has one central thesis:

> A computation can be understood not only as a state-transition system, but as a visible first-return behavior relative to an observer window.

Starting from that thesis, the paper developed five steps.

(1) Effective trace kernels give a machine-presentable model of computation.

(2) Full abstraction shows that trace order is exactly the preorder induced by finite visible observation.

(3) Trace transducers extend the framework to open systems and make it compositional.

(4) Trace complexity separates visible and hidden resource costs while recovering the classical classes.

(5) The proof theory turns local window structure and external refinement into a precise logic.

The running red–green–blue example illustrated the central intuition all the way through: one ambient process can induce several different visible computations, depending on which states are visible and which are hidden.

That is why trace theory is interesting. It says that computation is not only what a machine does internally. It is also what the machine *looks like* through a chosen interface of observation. The mathematics developed here shows that this idea is rich enough to support a full program in computation theory.

References

[1] D. Hoffman, C. Prakash, and S. Chattopadhyay. *Traces of Consciousness*. Draft manuscript, 2025.

[2] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42 (1936), 230–265.

[3] M. O. Rabin. Probabilistic automata. *Information and Control*, 6 (1963), 230–245.

[4] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. Van Nostrand, 1960.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.

[6] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119 (1996), 447–468.

[7] B. Bennett, D. Hoffman, and P. Murthy. Lebesgue logic for probabilistic reasoning and some applications to perception. *Journal of Mathematical Psychology*, 37 (1993), 63–103.

[8] J. Riddle and J. Schooler. Hierarchical consciousness: the Nested Observer Windows model. *Neuroscience of Consciousness*, 2024.

[9] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77 (1989), 257–286.

[10] M. L. Puterman. *Markov Decision Processes*. Wiley, 1994.