Simulation Learning: Agent-Guided Proof Search in Traced Monoidal Categories

Erik Arne Mathiesen-Dreyfus

Gimle Labs, Paris
www.gimlelabs.com

Abstract

The rewrite system of a traced monoidal category provides a sound proof system for circuit equivalence: two circuits connected by a chain of rewrites compute the same function. But many useful transformations—particularly approximate simplifications and trace elimination—have no purely syntactic proof. We propose *simulation learning*: an agent-based approach that extends the categorical proof system with simulation-guided reasoning, analogous to the weakening of pre- and postconditions in Hoare logic. The agent interleaves strict symbolic rewrites (which preserve exact equivalence) with simulation-informed approximate rewrites (which preserve behavioural equivalence up to a measured tolerance). We describe the framework, its connection to Hoare-style reasoning, and its application to discovering approximate closed-form solutions for systems where exact solutions do not exist.

1 Introduction

Consider the Navier–Stokes equations. They can be expressed as a system of differential equations, compiled into a traced circuit, and simulated. But no closed-form solution exists—there is no chain of categorical rewrites that eliminates the trace. The proof system is sound for exact equivalence, but the equations are not exactly solvable.

Yet physicists routinely work with approximate solutions: linearisations, perturbation expansions, asymptotic regimes, empirical correlations. Each of these is, informally, an *approximate rewrite*—a replacement of one subcircuit with a simpler one that behaves similarly under the conditions of interest. The approximation is justified not by a syntactic proof but by *simulation*: running both circuits and comparing their outputs.

This note proposes a framework for making such reasoning precise, by embedding it within the categorical proof system rather than outside it.

2 The Proof System and Its Limits

2.1 Exact rewrites

The traced monoidal category of circuits comes equipped with a rewrite system: categorical axioms (associativity, interchange, naturality of trace) and domain-specific rules (identity laws, scalar fusion, register–deregister cancellation). A chain of rewrites from circuit C_1 to circuit C_2 constitutes a *proof* that C_1 and C_2 compute the same function.

These proofs are *sound*: if a rewrite path exists, the circuits are truly equivalent. They are also useful: simplification, trace elimination, and closed-form discovery all reduce to finding rewrite paths. But the rewrite system has a fundamental limitation: it can only establish *exact* equivalence. Many practically important transformations—linearisation around an equilibrium, truncation of a series, replacement of a nonlinear term with an empirical approximation—are not exact.

2.2 The analogy with Hoare logic

In Hoare logic [4, 1], the rule of consequence allows weakening:

$$\frac{\{P'\}\ f\ \{Q'\} \qquad P \Rightarrow P' \qquad Q' \Rightarrow Q}{\{P\}\ f\ \{Q\}}$$

If the precondition can be strengthened or the postcondition weakened, the proof still goes through. This is what makes Hoare logic practical: you don't need to track exact state—you track *properties* of state, and the weakening rule lets you abstract.

We propose an analogous weakening for circuit equivalence. Instead of proving $C_1 = C_2$ (exact), we prove $C_1 \approx_\epsilon C_2$ (approximate to tolerance ϵ), where the approximation is *justified by simulation evidence*.

There is an important difference. In Hoare logic, the weakening steps ($P \Rightarrow P'$, $Q' \Rightarrow Q$) are *proven*—they have the same epistemic status as the rest of the proof. In our setting, approximate rewrites are justified by *simulation evidence*: finite samples from a computational experiment. This is a weaker form of justification. We adopt it because, for the systems we target, exact proof of the approximation's validity is typically as hard as solving the original problem. Simulation evidence is the best available justification when exact proof is out of reach, and the framework makes its use explicit rather than informal.

2.3 Approximate equivalence

We now give the approximate equivalence relation a precise definition.

Definition 1 (Approximate equivalence)**.** *Let $C_1, C_2 : m \to n$ be circuits with the same type, and let $D \subseteq \mathbb{R}^m$ be an input domain. We write $C_1 \approx_\epsilon^D C_2$ if*

$$\sup_{x \in D} \|C_1(x) - C_2(x)\| \leq \epsilon.$$

When the domain D is clear from context we write simply $C_1 \approx_\epsilon C_2$.

This is a pseudometric on circuits of the same type: $\approx_0$ is exact equivalence, and $\approx_\epsilon$ for $\epsilon > 0$ is a controlled relaxation. The key properties are:

Proposition 1 (Composition of approximate equivalences)**.** *If $C_1 \approx_{\epsilon_1}^D C_2$ and $C_2 \approx_{\epsilon_2}^D C_3$, then $C_1 \approx_{\epsilon_1+\epsilon_2}^D C_3$.*

Proof. Triangle inequality: $\|C_1(x) - C_3(x)\| \leq \|C_1(x) - C_2(x)\| + \|C_2(x) - C_3(x)\| \leq \epsilon_1 + \epsilon_2$. □

Proposition 2 (Monoidal product)**.** *If $f \approx_\epsilon^{D_1} f'$ and $g : m_2 \to n_2$ is any circuit, then $f \otimes g \approx_\epsilon^{D_1 \times D_2} f' \otimes g$ for any input domain $D_2 \subseteq \mathbb{R}^{m_2}$.*

Proof. $(f \otimes g)(x_1, x_2) = (f(x_1), g(x_2))$ and $(f' \otimes g)(x_1, x_2) = (f'(x_1), g(x_2))$. The difference is $(f(x_1) - f'(x_1), 0)$, which has norm $\|f(x_1) - f'(x_1)\| \leq \epsilon$. □

This is where the monoidal structure does real work: approximate rewrites on a subcircuit compose correctly with the rest of the system because the monoidal product preserves error bounds. The error does not leak across parallel wires.

Proposition 3 (Sequential composition and Lipschitz amplification)**.** *If $f \approx_\epsilon^D f'$ and $h : n \to p$ is L-Lipschitz on the range of f and f', then $h \circ f \approx_{L\epsilon}^D h \circ f'$.*

Proof. $\|h(f(x)) - h(f'(x))\| \leq L\|f(x) - f'(x)\| \leq L\epsilon$. □

This gives the subcircuit-to-whole-circuit error bound: if a subcircuit S is replaced by S' with error ϵ, and the surrounding context has Lipschitz constant L, the whole-circuit error is at most $L\epsilon$.

Proposition 4 (Error propagation in mixed proofs)**.** *Consider a mixed proof: a sequence of transformations $C_0 \to C_1 \to \cdots \to C_k$ where each step is either a strict rewrite (error 0) or an approximate rewrite with error ϵ_i. Let L_i be the Lipschitz constant of the composition of all subsequent strict rewrites after step i. Then*

$$C_0 \approx_E^D C_k \qquad \textit{where} \quad E = \sum_i \epsilon_i \prod_{j>i} L_j.$$

Proof. Induction on the proof length, applying Propositions 1 and 3 at each step. □

This quantifies the "cost" of a mixed proof: each approximate step contributes its error, amplified by the Lipschitz constants of all downstream transformations.

3 Simulation Learning

3.1 The framework

A *simulation-learning agent* operates on circuits in the sense of Asgard—typed morphisms in a traced monoidal category, compiled from equations and executed via differentiable simulation. The agent itself is realised within Hugin, an agent framework for interleaving symbolic and empirical reasoning. It uses two kinds of moves:

Definition 2 (Strict rewrite)**.** *A strict rewrite applies a categorical axiom or domain-specific rule to transform a circuit C into an exactly equivalent circuit C'. This is a syntactic operation that preserves semantics by construction.*

Definition 3 (Approximate rewrite)**.** *An approximate rewrite replaces a subcircuit S within C with a candidate replacement S', producing $C' = C[S \mapsto S']$. The replacement is justified by simulation evidence: the agent simulates both C and C' over a set of inputs and verifies that $\|C(x) - C'(x)\| < \epsilon$ for all tested inputs x.*

The agent interleaves these moves to search for a simplification of the original circuit. A typical strategy:

1. Apply strict rewrites to normalise and simplify where possible.
2. Identify a subcircuit S that is a candidate for approximation (e.g., a traced subnetwork, a nonlinear term).
3. Propose a simpler replacement S' (e.g., a trace-free circuit, a linearisation, a lower-order expansion).
4. Simulate both versions and measure the error.
5. If the error is within tolerance, accept the approximate rewrite and continue.
6. If not, revise the candidate or try a different subcircuit.

3.2 What counts as evidence

The simulation evidence for an approximate rewrite consists of:

- **Input distribution**: the set of inputs (initial conditions, parameter values, time horizons) over which the approximation was tested.
- **Error bound**: the maximum observed error $\epsilon = \max_x \|C(x) - C'(x)\|$ over the test inputs.
- **Confidence**: if the inputs are sampled stochastically, a statistical confidence bound on the error.

An approximate rewrite is *conditional*: it holds for the tested regime. The agent tracks the conditions under which each approximation was validated.

3.3 The proof object

A complete simplification by the agent produces a *mixed proof*: a sequence of strict rewrites (exact, unconditional) and approximate rewrites (simulation-justified, conditional). The result is a simplified circuit together with a record of:

- Which transformations are exact (and therefore hold universally).
- Which transformations are approximate (and the conditions under which they were validated).
- The overall error bound, composed from the individual approximate steps.

This is analogous to a Hoare logic proof with weakening steps: each weakening is justified, and the overall proof tracks the accumulated loss of precision.

3.4 Approximate rewrites as proof search heuristics

The mixed proof described above treats approximate rewrites as endpoints—accepted into the final proof when exact rewrites are unavailable. But there is a second, more subtle role: approximate rewrites can *guide* the search for strict rewrites, even when the final proof is entirely exact.

The key case is trace elimination. To eliminate a trace (feedback loop) from a circuit, one must exhibit a *generator*: an expression for the feedback wire that is consistent with the traced computation—in effect, a fixed point. This is analogous to finding a loop invariant in program verification, or a closed-form antiderivative in calculus: the result can be verified mechanically, but discovering it requires insight.

Approximate rewrites provide that insight. The agent can:

1. **Explore approximately**: use simulation to propose candidate generators. For instance, simulate the traced circuit, observe the sequence of values on the feedback wire, and fit an approximate expression (a truncated series, a rational function, an empirical formula).

2. **Refine**: use the approximate candidate as a starting point for exact search. If the candidate is $\hat{g}$, the agent searches for a strict rewrite path that establishes $g = \hat{g} + \delta$ where δ can be shown to vanish—or searches the neighbourhood of $\hat{g}$ in expression space for an exact generator.
3. **Verify strictly**: once an exact generator is found, the trace elimination proceeds by strict rewrite alone. The approximate reasoning was scaffolding; the final proof carries no approximation error.

This is a form of *proof sketching*: the approximate branch of reasoning produces a rough outline that the strict branch then fills in. The agent maintains two parallel threads—one approximate and exploratory, one strict and verifiable—and uses the former to steer the latter. The result is a strictly valid proof that the agent would not have discovered without the approximate detour.

Micro-example. Consider the traced circuit $\mathrm{Tr}(f)$ where $f : 1 \to 1$ is defined by $f(x) = ax + b$ with $|a| < 1$. The trace computes the fixed point of iteration $x_{n+1} = ax_n + b$, but to eliminate the trace we need to exhibit a generator g satisfying $g = ag + b$—that is, the fixed point in closed form.

The agent proceeds as follows. *Approximate branch*: simulate the iteration for several initial conditions and observe convergence to a value near $b/(1-a)$. Fit the approximate generator $\hat{g} = b/(1-a)$. *Strict branch*: substitute $g = b/(1-a)$ into the fixed-point equation and verify by strict rewrite: $a \cdot b/(1-a) + b = ab/(1-a) + b(1-a)/(1-a) = b/(1-a) = g$. The trace is eliminated exactly: $\mathrm{Tr}(f) = b/(1-a)$.

The approximate reasoning (simulation $\to$ curve fitting) discovered the candidate $b/(1-a)$; the strict reasoning (algebraic substitution) verified it. The final proof is exact. In a more complex setting—where the body of the trace is nonlinear or high-dimensional—the approximate branch narrows the search space from "all possible generators" to a neighbourhood of a simulation-validated candidate, making the strict search tractable.

4 Application: Approximate Closed-Form Solutions

The motivating application is discovering *approximate* closed-form solutions for systems that have no exact ones.

4.1 The task

Given a traced circuit C (representing a dynamical system), find a trace-free circuit C' such that $C \approx_\epsilon C'$ over a specified regime. The trace-free circuit C' is an approximate closed-form solution: it computes the system's behaviour without iteration, up to the tolerance ϵ.

4.2 Example: a nonlinear oscillator

Consider a damped nonlinear oscillator $\ddot{x} + \alpha\dot{x} + \beta x + \gamma x^3 = 0$. The cubic term prevents exact closed-form solution. An agent might proceed:

1. **Compile** the equation to a traced circuit with two feedback loops (position and velocity).
2. **Strict rewrite**: normalise the circuit, apply known identities.
3. **Simulate** over a range of initial conditions to understand the behavioural regime.
4. **Propose approximation**: replace the cubic subcircuit γx^3 with a linearised correction $\gamma' x$ where γ' is chosen to minimise simulation error over the tested regime. This is an approximate rewrite, justified by simulation.
5. **Strict rewrite**: with the cubic term removed, the circuit is now a linear second-order ODE, which *does* have a trace-elimination proof (the solution involves `exp` and `sin`/`cos` atomics).
6. **Eliminate trace**: apply the exact rewrites to obtain a trace-free circuit.
7. **Validate**: simulate the original and the trace-free approximation; record the error bound.

The result is an approximate closed-form solution: a trace-free circuit that matches the nonlinear oscillator's behaviour within tolerance ϵ for the tested regime.

4.3 Example: Navier–Stokes (approximate)

For the Navier–Stokes equations, the approximate pattern applies at a larger scale. The equations compile to a coupled traced circuit. No exact trace elimination exists. But the agent can:

- Identify regimes (low Reynolds number, boundary layers, far-field) where subcircuits simplify.

- Propose approximate rewrites justified by simulation in each regime.
- Compose regime-specific approximations into a piecewise closed-form solution.

  Each approximation carries its simulation evidence, making the conditions of validity explicit.

4.4 Remark: trace elimination and regularity

The proof-sketching technique also reframes certain existence problems. For the Navier–Stokes equations, the circuit C_{NS} is traced (temporal feedback), and global regularity amounts to the question: does a smooth generator for the trace exist? That is, can one exhibit $g = F(g)$ with g composed of smooth atomics and bounded derivatives? Simulation learning provides a search strategy: use approximate generators (fitted from numerical solutions) to constrain the search for strict generators, then verify by exact rewrite. The framework does not resolve the regularity question, but it makes the search *structured*—partial results (regularity for restricted initial data, bounded time horizons) are captured as valid mixed proofs with explicit approximation conditions.

5 Discussion

Soundness and validation. Strict rewrites are sound by construction: they preserve exact equivalence. Approximate rewrites are *empirically validated*: they hold for the tested inputs within the measured tolerance, but this is not soundness in the logical sense—it is a finite computational certificate. The overall proof is as strong as its weakest link, but crucially, the weakness is explicit, quantified, and localised to specific steps (Proposition 4).

Relationship to perturbation theory. Classical perturbation methods (regular, singular, multiple scales) are manual versions of simulation learning: the mathematician identifies a small parameter, proposes an approximate rewrite (expand in powers of ϵ), and validates by checking the residual. Our framework automates this and generalises it beyond expansion in a single small parameter.

Relationship to symbolic regression. Symbolic regression [6, 8] searches for mathematical expressions that fit data. Simulation learning is more structured: it searches within the space of circuits (typed, compositional, with a proof system), using simulation to justify specific rewrite steps rather than fitting an unconstrained expression. The result is not just an approximation but a *proof* of approximation, with explicit conditions of validity.

The role of the type system. The typed circuit representation constrains the search space: approximate rewrites must produce well-typed circuits (matching input/output degrees). This prevents the combinatorial explosion that plagues unconstrained symbolic search.

Implementation. The two sides of simulation learning map onto existing infrastructure. Asgard provides the circuit algebra: compilation from equations, categorical rewrites, and differentiable simulation under interchangeable calculi (real, stochastic, discrete). Hugin provides the agent side: structured reasoning over an immutable interaction stack, with tool access to Asgard's rewrite and simulation engines. The strict/approximate distinction is not merely conceptual—it corresponds to distinct tool invocations within the agent's action space.

Connection to reinforcement learning. The agent's search over mixed proofs can be formalised as an MDP: the state is the current circuit plus rewrite history; actions are strict rewrites (zero error cost) and approximate rewrites (incur error but unlock simplifications); reward is complexity reduction penalized by accumulated error. The agent learns when approximate moves pay off. This connects to the growing literature on learning to prove, where proof steps are selected by a learned policy [3, 7].

Related work. The notion of approximate behavioural equivalence has been studied in control theory as *approximate bisimulation* [2], where two dynamical systems are related if their trajectories remain within ϵ of each other. Our $\approx_\epsilon$ relation is analogous, lifted to the compositional setting of traced monoidal categories. In concurrency theory, simulation relations [5] provide a weaker-than-bisimulation notion of behavioural comparison; our approximate rewrites play a similar role, relaxing exact equivalence to tolerance-bounded agreement. The term "simulation learning" is chosen deliberately to evoke both senses: the agent learns *from* simulation (numerical evidence) and learns *simulation relations* (approximate behavioural correspondences). For proof search via RL in formal systems, see [3] (symbolic mathematics) and the AlphaProof line of work extending [7] to theorem proving.

Limitations. Several limitations deserve emphasis. First, simulation evidence is finite: an approximate rewrite validated over sampled inputs provides no guarantee for untested inputs. The error bound ϵ is empirical, not universal, and may underestimate worst-case behaviour. Second, error propagation through feedback is a concern: Proposition 4 gives a bound for feedforward mixed proofs, but when approximate rewrites occur inside a traced (feedback) subnetwork, errors can amplify exponentially with the number of iterations. Bounding this amplification requires Lipschitz analysis of the feedback body, which

may itself be intractable for the systems of interest. Third, there is an irreducible gap between statistical evidence and logical proof: a mixed proof with approximate steps is not a proof in the formal sense, and no amount of simulation can close this gap. However, we can use the framework of simulation learning to inform and improve the discovery of strict logical proofs.

6 Conclusion

Simulation learning extends the categorical proof system for circuit equivalence with simulation-justified approximate reasoning, analogous to the weakening rule in Hoare logic. This enables an agent to discover approximate closed-form solutions for dynamical systems that have no exact ones—not by abandoning formal reasoning, but by enriching it with empirical evidence. The result is a new kind of proof object: a mixed chain of exact rewrites and simulation-justified approximations, with explicit error bounds and conditions of validity.

The key insight is that simulation and proof are not in tension. Simulation provides the evidence that justifies approximate proof steps; the proof system provides the structure that keeps the approximations composable and their limitations transparent.

References

[1] Rob Arthan, Ursula Martin, Erik A. Mathiesen, and Paulo Oliva. A general framework for sound and complete floyd–hoare logics. *ACM Transactions on Computational Logic*, 11(1):7:1–7:31, 2009.

[2] Antoine Girard and George J. Pappas. Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control*, 13(2–3):214–228, 2007.

[3] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations (ICLR)*, 2020.

[4] Ursula Martin, Erik A. Mathiesen, and Paulo Oliva. Hoare logic in the abstract. In Zoltán Ésik, editor, *Computer Science Logic (CSL 2006)*, volume 4207 of *Lecture Notes in Computer Science*. Springer, 2006.

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

[6] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[7] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[8] Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.