26
AI for Mathematics

26.1 The Century of Formalization

In 1900, David Hilbert posed a challenge that defined the century: **Formalize all of mathematics.** He envisioned a system where truth could be determined mechanically. If one fed axioms into a machine, it would grind out all valid theorems. This dream directly inspired the invention of computers.

But Hilbert's dream hit two walls. First, the **Logical Wall** (1931). Kurt Gödel proved that Hilbert's machine was impossible. His *Incompleteness Theorems* showed that in any system rich enough to do arithmetic, there are true statements that cannot be proven. Truth is strictly larger than proof.

Second, the **Computational Wall** (1971). Even for statements that *can* be proven, finding the proof is hard. The P vs NP problem (Cook-Levin) formalized this intuition.

- **Verification is Easy (P):** If I give you a proof, you can check it in polynomial time.
- **Discovery is Hard (NP):** Finding that proof requires searching a space that grows exponentially.

For fifty years, this asymmetry was a burden. For Deep Learning, it is an opportunity. Because verification is cheap, we have an objective reward function. Because discovery is hard, we need a powerful heuristic to prune the search tree. Deep Learning is that heuristic. It replaces the brute-force search of the 20th century with the statistical intuition of the 21st.

26.2 The Infinite Board

Mathematics is the ultimate reinforcement learning environment. It offers infinite data, objective truth, and a sparse reward signal

that cannot be gamed. If you prove a theorem, you are right. If you hallucinate, you are wrong. There is no middle ground.

To understand the scale, consider the **Party Problem** (Ramsey Theory).

Theorem 26.2.1. *In any group of 6 people, there are either 3 mutual friends or 3 mutual strangers. (Formally, the Ramsey number* $R(3,3)$ *is* 6*).*

The Search Approach There are 2^{15} (32,768) possible relationship graphs. A brute-force solver checks them one by one. It is rigorous but blind. It learns nothing about the structure of friendship; it merely confirms the absence of a counter-example.

The Synthesis Approach A mathematician (or an advanced AI) does not check graphs. They apply the *Pigeonhole Principle*:

1. Pick Alice. She has 5 potential connections.
2. By the Pigeonhole Principle, she must have ≥ 3 friends OR ≥ 3 strangers.
3. Analyze only that subset. If it has even one pair of friends, we have found 3 mutual friends. If it does not, we have found 3 mutual strangers.

The proof collapses the search space from 32,768 graphs to 3 logical steps. This highlights how thinking of classic theorem proving as formal logic (or at least thinking about it as boolean satisfiability) is imbued with brute-force sensibility. By contrast, humans come up with problem-specific insight that shortens the search for the proof. AI for Math seeks to generate the *insight* that renders full search unnecessary. The hope is to capture human intuition and insight via a powerful LLM, and then leverage the promise of scaling to go beyond human capabilities.

26.3 The Formal Game: AlphaProof and Lean

To train an AI, we need a gym. **Lean** is that gym. To a computer scientist, Lean is not a text editor; it is a state machine. The "game" is to manipulate the state until the stack of goals is empty.

26.3.1 Lean as an RL Environment

Consider the proof that the sum of integers up to n is $n(n+1)/2$. We avoid integer division by proving $2\sum i = n(n+1)$.

Listing 26.1: A Simple Lean Proof

```
theorem gauss_sum (n : Nat) : 2 * sum_upto n = n * (n + 1) := by
  induction n with d hd
  -- Base Case
  . simp [sum_upto]
  -- Inductive Step
  . rw [sum_upto]
    rw [hd]
    ring
```

- **The State:** The prompt. Initially, it is the theorem statement.
- **The Action:** A tactic. The user types `induction n with d hd`.
- **The Transition:** The Lean Kernel executes the tactic. The state splits into two sub-goals: the Base Case ($n = 0$) and the Inductive Step ($n = d + 1$).
- **The Reward:** If the tactic is valid, the state updates. If invalid, the Kernel returns an error. The final reward (1.0) is given only when the state reads "No goals."

The critical feature here is the **Kernel**. It is a tiny, trusted piece of code (2000 lines) that verifies the logic. We do not trust the AI; we trust the Kernel. The AI is free to hallucinate wild tactics, because the Kernel acts as a perfect filter.

26.3.2 *AlphaProof: Small Model, Big Search*

Google's AlphaProof (IMO 2024 Silver Medalist) illustrates the modern architecture. It defies the "scale is all you need" trend. It uses a relatively small model (approx. 3B parameters) to drive a massive Monte Carlo Tree Search (MCTS).

The model provides the *policy* (which tactic to try next). The search provides the *rigor* (checking thousands of branches).

Heuristic 1: The Dual-Track Search (T vs $\neg T$) When the system is given a theorem T, it does not assume T is true. It spawns two parallel search processes:

1. **Track A:** Attempt to prove T.
2. **Track B:** Attempt to prove $\neg T$ (disprove T).

If Track B finds a counter-example, Track A is terminated immediately. This saves massive compute. It mimics the mathematician's skepticism: before trying to prove a conjecture, one tries to break it.

This is particularly vital in AI, where the formalization process is noisy. A valid English problem might be translated into a false Lean statement; the Dual-Track system detects this instantly.

Heuristic 2: Exhaust Mining (Autocurriculum) The bottleneck in formal math is data scarcity. There are not enough human-written proofs to train large models. AlphaProof solves this by mining its own failures.

When the system attempts a hard theorem and fails, it generates a massive search tree with thousands of intermediate sub-goals. Even if the main theorem is not proven, many sub-goals are. These are harvested. A failed attempt at a "Boss Level" problem yields successful training data for hundreds of lemma-level problems.

26.4 The Informal Game: Synthesis

Search is powerful, but it is memoryless. If a standard search hits a dead end, it backtracks and resets. It forgets *why* it failed. Large Language Models (LLMs) bring a new capability: **Constructive Backtracking**.

An LLM can look at k failed attempts and synthesize a solution that combines their partial successes.

The Process

1. **Attempt A:** The model tries proof by induction. It works for the algebraic manipulation but fails at the boundary condition.
2. **Attempt B:** The model tries proof by contradiction. It handles the boundary condition well but gets stuck in the algebra.
3. **Synthesis:** The model acts as a "Meta-Solver." It reads both traces. It synthesizes Attempt C: "Use the algebraic steps from A, but apply them within the contradiction framework of B."

This is not search; it is *Reasoning*. The model draws the map as it walks. It uses the debris of failure to construct the path to success.

26.4.1 Training Strategy: STaR

To teach this, we cannot just train on final answers. We must train on the process using the **Self-Taught Reasoner (STaR)** loop:

$$\text{Generate Solution} \rightarrow \text{Filter Correctness} \rightarrow \text{Fine-tune}$$

Crucially, we train on the *rationalization*—the chain of thought that led to the answer. The model learns to mimic the correction of errors, not just the avoidance of them.

26.5 Conclusion

Mathematics is the bridge between the statistical intuition of Deep Learning and the discrete rigor of Logic. By solving math, AI does not just learn to calculate; it learns to think. The P vs NP asymmetry, once a barrier, is now the engine of self-improvement. Verification is the teacher; the AI is the student; and the curriculum is infinite.