수학, 인공지능, 그리고 형식화 2 – 증명의 형식화와 인공지능

이시우 (UC Berkeley)

앞에서 우리는 인공지능을 어떻게 수학 연구에 활용할 수 있는지에 대해서 알아보았습니다. 이번 편에서는 최근에 떠오르고 있는 증명의 형식화(formalization)에 대해서 알아보고, AI, 특히 LLM 을 활용한 수학 연구에서의 형식화의 역할과 형식화에 있어서의 AI 의 역할과 주의할 점에 대해서 알아보면서 이 시리즈를 마치고자 합니다.

AI 가 증명을 했다고 주장하는데, 확실한가요?

앞서 언급했듯이 ChatGPT Pro 나 Gemini Deep Think 등 최전선에 있는 인공지능 모델들은 어느 정도 연구 레벨의 수학 정리를 증명할 수 있게 되었습니다. 하지만 그럴수록 더더욱 AI 가 생성한 증명을 회의적으로 바라볼 수 있어야 합니다. LLM 이 등장한 이후 수학 난제를 증명했다는 주장은 눈에 띄게 증가했고, 수학자들이 논문 초고를 업로드하는 아카이브(arXiv)에도 해당 분야 전문가가 보면 바로 "AI slop"임을 알 수 있는 증명들이 점점 늘어나고 있습니다. 점점 더 "그럴듯하게" 쓰인 증명들이 늘어나는 상황에서, 이를 검증할 수 있는 방법이 있으면 참 좋을 것 같지 않나요?

이를 해결하는 데 도움을 줄 수 있는 한 가지가 바로 형식 검증 언어(formal verification language)입니다. 형식 검증 언어란 수학적 명제와 그 증명을 컴퓨터가 이해할 수 있는 엄밀한 언어로 작성하고, 컴퓨터가 그 증명의 정확성을 기계적으로 확인할 수 있도록 설계된 언어입니다. 사람이 쓴 자연어 증명은 아무리 꼼꼼하더라도 논리적 비약이나 암묵적인 가정이 숨어 있을 수 있지만, 형식 검증 언어로 작성된 증명은 단 하나의 논리적 틈도 허용하지 않습니다. 이러한 특성 덕분에 형식 검증 언어는 수학뿐만 아니라 소프트웨어나 하드웨어의 정확성을 검증하는 데에도 널리 활용됩니다. 예를 들어 항공기 제어 시스템이나 반도체 설계처럼 오류가 허용되지 않는 분야에서는 이미 오래전부터 형식 검증이 중요한 역할을 해왔습니다. HOL, Isabelle, Coq/Rocq, Agda, Metamath, Mizar 등 여러 형식 언어들이 있지만, 그 중에서도 요즘 수학자들에게 가장 친숙한 언어는 아마 Lean[1]일 것입니다. Lean 은 Microsoft Research 에서 처음 개발되어 현재는 오픈소스로 운영되는 형식 검증 언어로, 수학자 친화적인 설계 덕분에 최근 수학 커뮤니티에서 빠르게 채택되고 있습니다.

Lean 으로 작성된 라이브러리 중 하나인 mathlib[2]은 수학에서 사용하는 여러 개념들과 정리들을 Lean 으로 형식화해서 모아놓은 일종의 디지털 도서관입니다. 현재 mathlib 에는 수만 개 이상의 정의와 정리가 형식화되어 있으며, 전 세계 수백 명의 수학자와 컴퓨터 과학자들이 지속적으로 기여하고 있습니다. 불과 5 년 전만 해도 mathlib 에 형식화된 정리들은 학부 수준의 지식이 대부분이었는데, 현재는 카테고리 이론, 대수적 정수론, 표현론, 편미분방정식, 대수적 위상수학 등 대학원 수준의 정리들도 어느 정도 형식화되어

[1] https://lean-lang.org/
[2] https://github.com/leanprover-community/mathlib4

있습니다. 특히 최신 연구 결과들도 어느 정도 커버할 수 있는 정도가 되었기 때문에, AI 가 갓 생성한 증명이 실제로 맞는지 Lean 으로 검증하는 것도 불가능하지만은 않은 일이 되었습니다.

자동 형식화 (Autoformalization)

Anthropic 의 Claude 나 OpenAI 의 Codex 는 이제 대부분의 개발자에게 거의 일상이 되었습니다. 한 번에 몇 천~몇 만 줄의 코드를 작성할 수 있는 이런 AI 들을 이용하면, 코딩을 배우지 않은 초보자도 며칠 만에 앱을 뚝딱 만들어 낼 수 있습니다. 이러한 관점에서 코딩 AI 들을 수학 증명 형식화에 사용하는 것은 아주 자연스러운 아이디어이고, 실제로 "어느 정도는" 잘 작동하는 것을 확인할 수 있습니다. 하지만 이러한 LLM 기반의 코딩 AI 들은 인터넷에 퍼져 있는 여러 코드들을 기반으로 훈련되었기 때문에, 당연히 Python 이나 Java 같이 많이 사용되는 언어에서는 높은 성능을 보이는 반면, 비교적 적은 사람들이 사용하는 Lean 이나 다른 형식화 언어에서는 그에 미치지 못하는 성능을 보이기 마련입니다[3]. 그래서 특정 언어에 특화된 AI 를 만드는 방향을 생각할 수 있고, Axiom Math, Harmonic, Logical Intelligence, Math Inc 등의 기업들이 각자의 AI 를 열심히 개발하고 있습니다.

Harmonic 에서는 Aristotle 이라는 자동 형식화 모델을 개발 중입니다[4]. 누구나 사용할 수 있도록 공개되어 있어 많은 수학자들이 여러 정리의 형식화에 Aristotle 을 활용하고 있습니다. Aristotle 은 작년 IMO 문제들에 대해 DeepMind 와 마찬가지로 금메달에 해당하는 성적을 얻었을 뿐만 아니라, 푼 문제들에 대한 모든 증명을 형식화하는 데에도 성공했습니다. 또한 최근 AI 의 도움으로 해결된 에르되시 문제의 대부분이 Aristotle 의 도움으로 형식화되었고, 수학 뿐만 아니라 코드 검증 벤치마크인 VERINA 에서도 괄목할 만한 성능을 보여주었습니다[5].

Math Inc 는 자동 형식화 모델 Gauss 를 만드는 또다른 기업으로, 작년에 ABC 추측의 약한 버전에 대한 짧은 증명을 자동 형식화 한 데에 이어[6], 소수 정리와 유한 체 위의 곡선에 대한 리만 가설의 증명의 자동 형식화에 성공했습니다[7]. 더 나아가서, 제가 2 년 전부터 관리자 중 한명으로 참여하고 있는 Maryna Viazovska 교수님의 8 차원 공쌓기 문제 증명의 형식화 프로젝트[8]의 남아있던 부분들을 자동 형식화 하고, Henry Cohn, Abhinav Kumar, Stephen D. Miller, Danylo Radchenko, Maryna Viazovska 교수님의 24 차원 공쌓기 문제의 증명 역시도 자동 형식화를 해냈습니다[9]. (이에 대해서는 뒤에서 좀 더 자세히 이야기할 예정입니다.)

예시: 교차수 최적화 문제, 부분 정규 소수 추측

---

[3] 물론 항상 그렇다는 뜻은 아닙니다. 사실 Claude 나 Codex 도 Lean 에 대한 추가적인 설정 없이도 꽤 많은 경우에 돌아가는 Lean 코드를 작성할 수 있습니다.

[4] Aristotle 홈페이지: https://aristotle.harmonic.fun/

[5] Press release: https://harmonic.fun/news/verina-benchmark/

[6] GitHub: https://github.com/morph-labs/lean-abc-true-almost-always , 그 당시의 이름은 Trinity 였습니다.

[7] GitHub: https://github.com/math-inc/strongpnt, https://github.com/math-inc/RiemannHypothesisCurves

[8] GitHub: https://github.com/thefundamentaltheor3m/Sphere-Packing-Lean

[9] Press release: https://www.math.inc/sphere-packing

실제로 자동 형식화 AI 들을 잘 이용해서 수학 연구를 좀 더 효율적으로 한 두 가지 예시를 소개하고자 합니다.

ETH Zurich 의 Johannes Schmitt 연구원[10]은 여러 연구자들과 함께 AI 를 위한 연구 레벨의 수학 데이터셋인 ImProofBench[11]를 만들고 있습니다. 이 과정에서 대수기하학에 등장하는 특정 교차수(intersection number)의 최댓값과 최솟값을 구하는 다음과 같은 문제를 생각하게 되었습니다.

음이 아닌 정수 g, n 에 대해서 $2g – 2 + n > 0$ 이라고 하고, 종수(genus) 가 g 이고 n 개의 표시점이 있는 안정 곡선(stable curve) 들의 모듈라이 공간(moduli space) 를 $\overline{\mathcal{M}}_{g, n}$ 라고 하자. 각각의 표시점 i 에 대해, 그 점에서의 공접다발(cotangent bundle) 의 첫번째 천 특성류(Chern class) 를 ψ_i 라고 하자. 이 때, n 개의 음이 아닌 정수 $e_1, \dots, e_n$ 에 대해, ψ-class 의 교차수를

*$$
D(e_1, \dots, e_n) = \langle \tau_{e_1} \cdots \tau_{e_n} \rangle_g = \int_{\overline{\mathcal{m}}_{g, n}} \psi_1^{e_1} \cdots \psi_n^{e_n}
$$*

이라고 정의하고, 이는 항상 유리수이면서 $e_1 + \cdots + e_n = 3g – 3 + n$ 이 아닌 $e_1, \dots, e_n$ 에 대해서는 0 이 된다. 이 때, 단체(simplex)

*$$
E(g, n) := \left\{\mathbf{e} = (e_1, \dots, e_n) \in \mathbb{Z}_{\ge 0}^n : \sum_{i=1}^{n} e_i = 3g – 3 + n\right\}
$$*

위에서 D 의 최솟값과 최댓값은 무엇이며, 언제 최댓값과 최솟값을 갖는가?

위에 나온 용어를 모두 이해할 필요는 없습니다[12]. 특정 조건을 만족하는 정수점들의 집합 위에서 정의된 함수가 있고, 이 함수가 언제 최댓값과 최솟값을 가지는지에 대한 문제 정도로 이해하시면 됩니다. Schmitt 연구원은 이 문제가 떠올리기에는 자연스러웠지만 실제로는 많이 연구되지 않은 종류의 새로운 문제임을 알게 됩니다. 정확한 답을 모르는 상태에서 AlphaEvolve 의 오픈소스 재구현 버전인 , OpenEvolve[13]를 통해 이 문제에 대한 답을 추측할 수 있었고, 이후 이 문제를 ImProofBench 에 추가해 여러 LLM 으로 테스트해 본 결과 실제로 간단한 답을 얻을 수 있었다고 합니다.

좀 더 자세히 살펴보면, GPT-5 가 제안한 풀이는 크게 두 단계로 이루어집니다[14]. 먼저, $E(g, n)$ 과 같은 단체 위에서 정의된 양의 유리수 값을 가지는 함수가 대칭성(symmetry)과 로그-볼록성(log-concavity)을 만족한다면, 이 함수의 최댓값은 균형 벡터(balanced vector)에서, 최솟값은 집중

[10] Website: https://johannesschmitt.gitlab.io/
[11] Website: https://improofbench.math.ethz.ch/
[12] 저도 사실 잘 모릅니다.
[13] GitHub: https://github.com/algorithmicsuperintelligence/openevolve
[14] arXiv: https://arxiv.org/abs/2512.14575

벡터(concentrated vector)에서 달성됨을 보입니다(논문의 Theorem 3.1). 그런 다음, 위에서 정의된 교차수가 이러한 성질들을 만족함을 보임으로써 증명이 완성됩니다(논문의 Proposition 3.2).

그런데 GPT-5 의 증명이 맞다는 것을 어떻게 확신할 수 있을까요? 지금까지 소개한 내용들로 미루어 보면, 형식화를 했다는 것을 예상하셨을 수도 있습니다. 실제로 Theorem 3.1 은 형식화하기에 비교적 쉬운 정리로, Claude Code 와 ChatGPT 의 도움으로 형식화할 수 있었다고 합니다[15]. 다만 Proposition 3.2 는 Schmitt 연구원이 직접 증명했고, 아직 Mathlib 에 없어 형식화하기 어려운 대수기하학의 여러 개념과 결과들을 사용하기 때문에 형식화는 하지 않았습니다.

또 다른 예시로, Axiom Math[16]에서는 AxiomProver 라는 주어진 수학 문제를 자동으로 풀고 Lean 으로 검증까지 하는 자동 추론 모델(automated reasoning model)을 개발 중이며, 이 모델을 이용해서 여러 수학 연구 문제들에서 진전을 이루었습니다[17]. 그 중 정수론의 유명한 추측 중 하나인 Kummer-Vandiver 추측과 관련된 결과를 간략히 소개하고자 합니다. 주어진 소수 p가 2부터 $p - 3$까지의 모든 짝수 $2k$에 대해서 베르누이 수(Bernoulli number) B_{2k}의 분자를 나누지 않으면 p를 정규 소수(regular prime)라고 합니다. 정규 소수가 무한히 많은지는 아직까지도 미해결 추측으로 남아 있는데, AxiomProver 는 다음과 같은 이 추측의 약한 버전을 증명했습니다.

각각의 자연수 m과 $2 \le 2k \le \min\{m, p – 3\}$에 대해서 p가 B_{2k}의 분자를 나누지 않으면 p를 m-regular 라고 하자. 실수 $\alpha > 1/2$와 소수 p에 대해서 $M_\alpha(p) := \lfloor \frac{\sqrt{p}}{(\log p)^\alpha} \rfloor$라고 하자. 이때 X까지의 소수 중에서 $M_\alpha(p)$-regular 가 아닌 소수의 갯수는 $C_\alpha \frac{X}{(\log X)^{2\alpha}}$ 이하이다.

소수정리 (prime number theorem)에 의하면 X까지의 소수의 개수는 근사적으로 $\frac{X}{\log X}$개라는 것이 알려져 있기 때문에, 위 결과의 따름정리로서 대부분의 소수는 $M_\alpha(p)$-regular 라는 사실을 얻게 됩니다. AxiomProver 는 위의 정리의 statement 만 가지고 추가적인 도움 없이 증명을 완성했고, 나아가 Lean 으로 형식화까지 할 수 있었습니다(그렇기 때문에 증명이 맞다고 확신할 수 있습니다). 증명 자체는 비교적 어렵지 않지만, AI 가 연구 레벨의 문제를 사람의 개입 없이 풀고 형식화까지 해낸다는 것은 꽤 고무적인 결과라고 생각합니다.

자동 형식화의 함정

앞에서 소개한 여러 예시들을 종합하면, 인공지능을 활용한 수학 연구의 한 가지 방향을 다음과 같이 정리할 수 있습니다.

1. LLM 을 이용해서 새로운 수학 정리에 대한 "증명"을 얻습니다.

[15] GitHub: https://github.com/schmittj/balanced-vectors-blueprint
[16] https://axiommath.ai/
[17] https://axiommath.ai/papers

2. "증명"이 정말로 옳다는 것을 확인하기 위해 증명을 형식화를 합니다. 이 과정에서도 LLM 을 활용할 수 있습니다.
3. 컴파일이 되는 Lean 코드를 작성하고 나면 증명에 대한 100% 확신을 얻을 수 있습니다.

이미 앞에서 이 사이클에 해당하는 몇 가지 예시를 소개했고, 앞으로 점점 더 이런 방식의 연구가 많아질 것이라 생각합니다. 하지만 이게 과연 올바른 프로세스일까요?

과정만 놓고 보면, 컴파일이 되는 Lean 코드를 작성하면서 검증까지 마치기 때문에 LLM 을 활용하더라도 아무런 문제가 없다고 생각할 수 있습니다. 하지만 여기에는 많은 사람들이 간과하는 부분이 있는데, 바로 **컴파일이 되는 Lean 코드가 증명한 명제와 실제로 의도한 명제가 다를 수 있다는 점**입니다. 예를 들어, 아래의 Lean 코드는 "HyperTree Proof Search for Neural Theorem Proving"이라는 논문에 등장하는 IMO 문제 하나를 자동으로 형식화 한 결과입니다.[18]

```
theorem imo_2001_p6
    (a b c d : Nat)
    (h0 : 0 < a ∧ 0 < b ∧ 0 < c ∧ 0 < d)
    (h1 : d < c)
    (h2 : c < b)
    (h3 : b < a)
    (h4 : a * c + b * d = (b + d + a - c) * (b + d - a + c)) :
    ¬ Nat.Prime (a * b + c * d) := by
  contrapose h4
  rw [mul_comm]
  simp [Nat.prime_def_lt, not_le_of_gt h0.1, not_forall, not_le_of_gt h3,
    Nat.mul_sub_right_distrib, Nat.add_comm]
  contrapose! h4
  contrapose! h4
  apply LT.lt.ne
  apply Nat.lt_sub_of_add_lt
  nlinarith
```

원래 문제는 양의 정수 $a > b > c > d$가 $ac + bd = (b + d + a – c) (b + d – a + c)$를 만족하면 $ab + cd$가 합성수임을 보이는 것이고, 얼핏 잘 형식화된 것처럼 보입니다. 하지만 이는 잘못된 형식화입니다.

---

[18] NeurIPS 2022, arXiv https://arxiv.org/abs/2205.11491 Appendix 의 마지막 예시입니다. 원래 논문에서는 Lean 3 로 작성되어 있는데, 본문에서는 이를 Lean 4 문법으로 재 작성했습니다. https://live.lean-lang.org/ 에서 실제로 확인해볼 수 있는데, 다음 링크에 있는 코드를 복사 & 붙여넣기 하시면 됩니다. https://gist.github.com/seewoo5/a387e455482159f61b34763749b6d047

Lean 에서의 자연수의 뺄셈에서는 작은 수에서 큰 수를 빼면 0 이 되도록 정의하고 있기 때문에[19], h4 의 $b + d – a + c$는 수학적으로는 $\max\{(b + d – a), 0\} + c$가 되어서, 아예 다른, 실제로 더 쉬운 문제가 됩니다[20].

그렇다면 명제만 확인하면 괜찮은 걸까요? 그것도 아닙니다. 우리의 목적이 어떤 "증명"이 올바른지를 Lean 형식화를 통해 검증하는 것이라고 합시다. 소수가 무한히 많다는 명제의 가장 유명한 증명은 다음과 같은 유클리드의 증명입니다.

1. 소수가 유한히 많다고 가정하고, 이를 $p_1, \dots, p_m$ 이라고 합시다.
2. $N = p_1 p_2 \cdots p_m + 1$이라는 수를 정의합니다. 그러면 이는 1 보다 크고, 1 보다 큰 자연수는 항상 어떤 소수에 의해서 나누어 떨어져야 하기 때문에 이를 나누는 소수 p가 존재합니다.
3. 하지만, $p_1, \dots, p_m$ 중 어떤 것도 N을 나누면 1 이 남기 때문에, p는 $p_1, \dots, p_m$과 다른 새로운 소수가 되어야 해서 모순이 됩니다.

이제 이 증명을 AI 에게 형식화를 맡겼다고 하고, 다음의 코드를 받았다고 생각해봅시다.

```
theorem infinitude_of_primes : {p : ℕ | p.Prime}.Infinite := by
  intro hfin
  obtain ⟨n, hn⟩ := hfin.bddAbove
  have hne : n.factorial + 1 ≠ 1 := by
    intro h; exact Nat.factorial_ne_zero n (by omega)
  obtain ⟨p, hp, hpdvd⟩ := Nat.exists_prime_and_dvd hne
  exact hp.not_dvd_one <| by
    have hpnfac : p ∣ n.factorial := (Nat.Prime.dvd_factorial hp).2 (hn hp)
    simpa using Nat.dvd_sub hpdvd hpnfac
```

이 Lean 코드는 컴파일이 잘 되는 Lean 코드이고, statement 역시 자연수 중에서 소수인 것이 무한히 많다라는 주장을 잘 형식화 한 것을 알 수 있습니다. 그런데, 증명 그 어디에도 $N = p_1 p_2 \cdots p_m + 1$은 찾아볼 수 없고, 대신 $n! + 1$ (`n.factorial + 1`)을 발견할 수 있습니다. 이 증명을 다시 자연어로 풀어서 쓰면 다음과 같습니다.

1. 소수가 유한히 많다고 가정하고, 그 소수들보다 크거나 같은 n을 하나 고릅니다.
2. $N = n! + 1$을 생각하면, 이는 항상 1 이 아니기 때문에 이를 나누는 소수 p가 존재해야 합니다.
3. 그런데 그러한 소수는 $n!$역시 나눠야 하기 때문에, 그 차이인 1 도 나눠야 해서 모순이 됩니다.

비슷하지만 사용하는 N이 다른 약간 다른 증명입니다. 앞서 언급한 유클리드의 증명을 더 충실하게 따르는 형식화된 증명은 다음과 같습니다(증명하고자 하는 명제는 동일합니다).

---

[19] 왜 Lean 에서 이렇게 정의하고 있는지에 대해서는 Kevin Buzzard 의 블로그에서 잘 설명하고 있습니다. https://xenaproject.wordpress.com/2020/07/05/division-by-zero-in-type-theory-a-faq

[20] 실제로 위 코드를 컴파일해보면, 몇몇 가정들은 증명에서 사용되고 있지 않고, 증명에서도 필요 없는 부분들이 있는 것을 확인할 수 있습니다.

```
theorem infinitude_of_primes' : {p : ℕ | p.Prime}.Infinite := by
  intro hfin
  have hne : hfin.toFinset.prod id + 1 ≠ 1 := by
    have : 0 < hfin.toFinset.prod id := Finset.prod_pos fun i hi =>
      (hfin.mem_toFinset.mp hi).pos
    omega
  obtain ⟨p, hp, hpdvd⟩ := Nat.exists_prime_and_dvd hne
  have hpS : p ∣ hfin.toFinset.prod id :=
    Finset.dvd_prod_of_mem id (hfin.mem_toFinset.mpr hp)
  exact hp.not_dvd_one (by simpa using Nat.dvd_sub hpdvd hpS)
```

아주 짧게 설명하자면, `hfin` 이 소수가 유한히 많다고 가정하는 부분이고, `hfin.toFinset.prod id + 1` 가 (소수들의 곱) + 1 에 해당합니다. `Nat.exists_prime_and_dvd`은 1 이 아닌 자연수를 나누는 소수가 항상 존재한다는 정리이고, 그 소수는 (소수들의 곱) 역시 나눠야 하기 때문에 (hpS), 그 차이인 1 도 나눠야 해서 (`Nat.dvd_sub`) 모순이 됩니다. 두 증명이 언제 같고 다른지는 사실 굉장히 답하기 어려운 질문인데, Gowers 교수님의 블로그[21]와 MathOverflow 의 유저 Martyguy 의 질문[22]에 대한 여러 답변들을 참고하시면 좋을 것 같습니다.

위의 예시들은 모두 코드가 짧기 때문에 이러한 차이점을 발견하는 것이 어렵지 않습니다. 하지만, 코드가 매우 길다면 어떨까요? 앞서 언급했듯이 Math Inc 의 Gauss 는 8 차원과 24 차원 공 쌓기 증명의 자동 형식화를 완료 했을 당시에 프로젝트의 60~70%가 이미 진행된 상태에서 저희가 2 년간 작성한 코드는 약 17000 줄이었습니다. 그리고 Gauss 가 나머지 30~40%를 형식화한 코드는 약 50000 줄로 예상을 훨씬 웃도는 수준이었습니다. 24 차원의 경우는 약 180000 줄로, 초기의 500000 줄에서 많이 줄인 결과라고 합니다. 이 정도 규모의 코드를 한 줄씩 확인하는 것은 사실상 불가능에 가깝고, 정의나 정리가 의도한 대로 형식화되었는지 확인하는 것만 해도 상당한 노력을 필요로 합니다.

더 중요한 것은 프로젝트의 목적입니다. Viazovska 교수는 이 증명으로 2022 년 필즈상을 받았고[23], 증명에 문제가 있다고 생각하는 사람은 아무도 없습니다. 저희가 이 프로젝트를 시작한 이유는 증명의 검증이 아니었습니다. 실제 목표는 다른 프로젝트에서도 재사용할 수 있는 정의와 정리들을 최대한 간결한 코드로 작성하고, 직접 형식화를 하면서 증명 자체를 더 깊이 이해하는 것이었습니다. 실제로 가중치가 2 인 아이젠슈타인 급수(Eisenstein series)나 모듈러 형식(modular form)의 미분, 그리고 norm_numI 나 tendsto_cont 와 같은 새로운 전략(tactic)들이 프로젝트를 진행하는 과정에서 자연스럽게 나왔고, 이는 다른 프로젝트에서도 사용할 수 있도록 mathlib 에 추가하는 중 입니다. 이러한 관점에서 보면, Math Inc 의 자동 형식화를 통해 어쩌면 얻은 것보다 잃은 것이 더 많을지도 모릅니다.[24]

---

[21] https://gowers.wordpress.com/2007/10/04/when-are-two-proofs-essentially-the-same

[22] https://mathoverflow.net/questions/3776/when-are-two-proofs-of-the-same-theorem-really-different-proofs

[23] 이철희 교수님의 Horizon 글: https://horizon.kias.re.kr/24344/

[24] Jeremy Avigad 교수의 글에서 더 자세한 이야기를 찾아보실 수 있습니다. https://arxiv.org/abs/2603.03684

요약하자면, 컴파일이 되는 Lean 코드가 알려주는 것은 “형식화된 명제가 참이다” 라는 사실 뿐입니다. 만약 자동 형식화로부터 내리고 싶은 결론이 “가지고 있는 자연어 명제와 그에 대한 증명이 참이다” 라면,

1. 명제가 제대로 Lean 으로 형식화가 되었는지를 확인해야 하고,
2. Lean 증명 역시 자연어로 된 증명의 논리를 따르고 있는지 확인해야 합니다.

2 번과 관련해서는 Simon Dedeo, Eamon Duede 의 “A correspondence problem for mathematical proof”라는 최근 논문을 참고하시기를 바랍니다[25]. 자동 형식화와 관련된 상당수의 연구들이 이 두 가지를 놓치고 있습니다. 특히, LLM 기반의 자동 형식화 인공지능이 발전하면서 점점 더 길고 읽기 어려운 코드를 작성하고 있는데, 이를 실제로 읽지 않는다면 아무런 의미가 없습니다. 만약 누군가가 밀레니엄 난제 중 하나를 인공지능으로 풀었다고 주장하고 형식화까지 했다고 하더라도, 실제 Lean 코드가 1 억 줄이라면 수학자가 그로부터 얻을 수 있는 것이 과연 있을까요? 수학자들은 단순히 명제의 참과 거짓에 관심이 있는 것이 아니라, 그 증명 자체에 더 큰 관심이 있습니다. 읽을 수 없는 증명은, 설령 그것이 형식적으로 검증되었다 하더라도, 수학자들에게는 아무런 의미를 갖지 못합니다.

먼저 온 미래

이 글에서는 AI 가 최근 수학 연구에 실제로 어떻게 활용되고 있는지 살펴보았습니다. 그런데 수학 연구를 잘하는 AI 가 등장한다면, 이것이 꼭 좋은 일일까요? 박사 졸업을 목전에 둔 대학원생으로서, AI 가 언젠가 저보다 수학을 훨씬 더 잘하게 되면 어쩌나 하는 고민을 요즘 들어 자주 하게 됩니다. AI 가 계속 발전해서 박사 논문 수준의 연구를 10 분 만에 끝내고 난제를 하루 만에 풀어버린다면, 수학자의 역할은 앞으로 어떻게 될까요?

최근에 장강명 작가의 <먼저 온 미래>를 재미있게 읽었습니다. 알파고 이전과 이후의 바둑을 통해 현재 AI 의 발전 상황을 설명하는 책인데, 요즘 바둑 기사들은 거의 당연하게 AI 를 활용해 바둑을 공부한다는 사실을 알게 되었습니다. 그렇지 않으면 AI 를 공부한 다른 기사를 이기는 것이 불가능해졌기 때문이라고 합니다. 처음 50 수 정도는 "AI 추천수"를 외워서 두기 때문에 바둑의 초반 포석이 비슷해졌다는 평가도 있습니다. 현재 세계 1 위인 신진서 기사는 AI 를 특히 잘 활용하는 선수로도 유명하다고 합니다.

수학 연구도 비슷한 길을 따라갈지 모릅니다. 점점 더 많은 수학자들이 AI 를 연구에 활용하고 있고, 가까운 미래에는 AI 를 활용하지 않는 것이 오히려 이상하게 여겨지는 때가 올 것 같습니다. 다만 수학과 바둑 사이에는 중요한 차이가 있습니다. 수학자의 가장 큰 목적은 다른 수학자와의 경쟁이 아니라 수학 그 자체에 있다는 것입니다[26]. 수학자들이 AI 를 연구에 활용하는 이유는 다른 수학자보다 더 많은 논문을 쓰기 위해서가 아닙니다. 인간이 하기 어려운 부분에서 AI 의 도움을 받아 수학적 진리에 더 빠르게 다가가는 것,

---

[25] arXiv: https://arxiv.org/abs/2603.13680

[26] 물론 바둑도 바둑 그 자체에 목적을 두는 기사분들도 많이 있다고 생각하고, 이 역시 앞서 언급한 <먼저 온 미래>에도 나와있습니다. 전 바둑 기사였던 이세돌님은 알파고 이후로 바둑이 더이상 예술이 되지 않은 점에 대해서 실망했다고 합니다

그 도구이자 공동 연구자로서 AI 를 활용하는 것이 올바른 방향이라고 생각합니다. 가끔 AI 로 "수학 전체를 풀겠다"는 목표를 내세우는 기업들을 보게 되는데, 저는 수학이 아름다운 이유가 인간이 풀기에는 너무나 방대하고 그만큼 탐구할 것이 많기 때문이라고 생각합니다. 수학의 모든 것을 해결하는 것은 불가능하겠지만, 그 탐구의 속도를 높이는 데 AI 가 점점 더 큰 역할을 하게 될 것이라 기대합니다.

(이 시리즈의 두 글을 문법적으로 첨삭, 교정하는 과정에서 Anthropic 의 Claude 를 사용하였고, 위의 소수의 무한성 증명에 관한 Lean 코드 작성에는 OpenAI 의 Codex 와 Anthropic 의 Claude 가 사용되었습니다.)