증명을 코딩하기

기계적으로 증명이 맞다는 걸 확인하는 방법

KAIST Concurrency & Parallelism Laboratory 이장건

증명을 확신하는 방법

흔한 학부생이 증명을 하는 과정

증명을 확신하는 방법

- 시험이나 과제에서 증명 문제가 나옴.
- 열심히 공부해서 배운 정리들과 mathoverflow의 힘을 합쳐서 증명함.
- 맞다는 확신을 가지고 제출하고 놀러감.
- 다시 받은 점수는 처참하게 까여있음.
  - ‘사용한 정리의 가정이 만족하지 않아요.’
  - ‘증명의 논리적 비약이 너무 심해요.’
  - ‘사용한 보조 정리의 증명을 안 하셨어요. 간단하지만 해야 해요.’

왜 이런 일이 일어날까?

증명을 확신하는 방법

- 수학이 문제일까?
  - 수학이 옳다고 가정하고 증명을 하는 것.
- 사람이 문제일까?
  - 기억력 짧음. 실수 자주 함. 틀린 게 맞다 생각함 등…
  - 증명이 맞다 확신하는 것이 증명한 본인임.
  - 조교 또한 사람이니 실수했을 수도 있음
- 증명을 하는 사람과 독립적으로 증명이 옳다는 확신을 얻을 수 있을까?

증명이란 무엇일까?

증명을 확신하는 방법

- ‘정리가 옳다고 확신하는 논리적인 주장.’
- 좀 더 엄밀하게 생각해보자.
  - 형태: $A \to B$.
  - 목표: A가 옳다고 가정하고, B의 올바름을 보이기.
  - 방식: A를 공리를 사용해서 변환, 이미 증명한 정리를 통해 변환.
- 이러한 단계를 기계의 도움을 받아 할 수 있을까?

Proof Assistant

증명을 확신하는 방법

- **목표:** 증명의 논리적인 단계들을 올바르게 했음을 검증.
  - 공리를 통한 변환.
  - 이미 증명된 정리를 통한 변환.
    - 정말 정리를 적용할 수 있는지 확인.
    - 정리의 조건들을 만족하는지 확인.
  - 증명을 까먹고 안 한 것이 없는지 확인.
- **장점:** 증명 완성 = 증명의 진행의 엄밀한 과정이 완성.

예시 1: 간단한 정리의 증명

증명을 확신하는 방법

- 정리: 모든 n에 대해, $n + 0 = n$.
- 증명: 다음 두 요소에 대해 귀납을 진행.
  - 자연수의 귀납 정의 (0, S 0, S (S 0), …).
  - 덧셈의 귀납 정의 (0 + n = n. S n + m = S (n + m)).
- 실제 코드로 봐보자.

예시 2: 틀린 증명이 틀리다는걸 확인

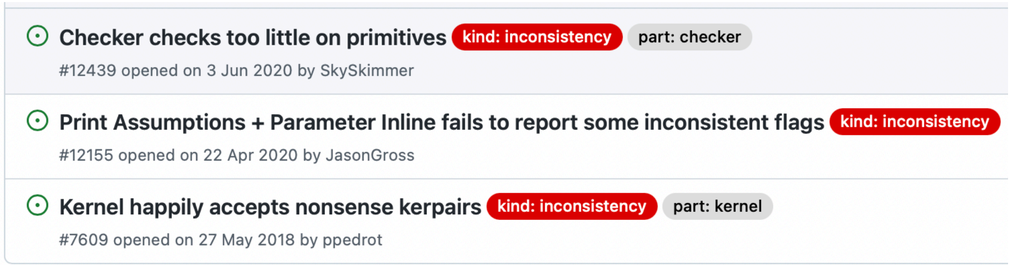
증명을 확신하는 방법

- 정리: ‘모든 사람은 같은 키다’.
- 증명: 집합 크기에 대한 귀납법.
  - N = 0,1: 공집합 및 사람이 혼자임으로 자명.
  - N > 1: 집합 P안의 사람들을 $p_{1,...}p_n$이라 하자.
    - P에서 p_1을 제거한 집합을 P_1, p_n을 제거한 집합을 P_2라 하자.
    - 그럼 P_1, P_2의 크기는 P보다 작으므로, 귀납법에 의해서 같은 키의 사람만 있다.
    - $P_1 \cap P_2$는 두 집합 모두에 있는 사람이 있으므로, 모든 사람의 키는 같다.
- $N = 2$일때 마지막 문장 성립 X.

Q: 그럼 Proof Assistant로 검증된 증명은 맞나요?

증명을 확신하는 방법

- 아니요!
- ‘대충 False가 증명 가능하다는 내용’.
- 원래 신뢰의 대상: 증명을 진행한 **다양한 사람들.**
- 새로운 신뢰의 대상: 증명을 확인하는 **Proof Assistant의 구현.**
- 임의의 불확실한 신뢰 대상 → 공통된 기계적인 신뢰 대상.



수학에서 Proof Assistant의 사용

왜 수학에서 써야 할까?

수학에서 Proof Assistant의 사용

- 어떠한 증명이 올바르다는걸 확신해야 할 이유가 있음.
  - 증명했지만, 리뷰를 기다려야 한다.
  - 증명했지만, 다른 사람이 받아들이기 어려운 컴퓨터의 도움을 받았다.
  - 증명했지만, 다른 사람들을 납득시키기 어렵다.
  - 증명을 이해했지만, 모든 내용을 이해했는지 모르겠다.
- Proof Assistant는 이러한 문제들을 해결할 수 있음.

Four Color Theorem (1/2)

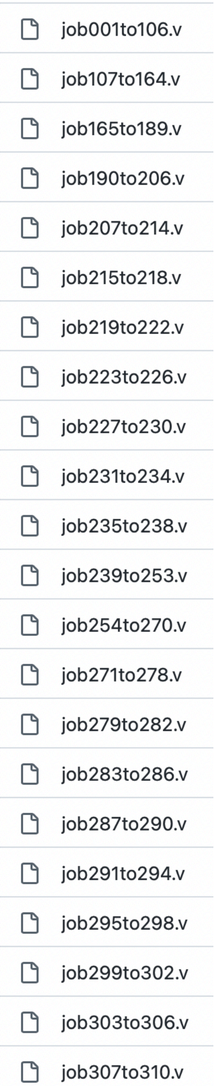
수학에서 Proof Assistant의 사용

- 어떠한 지도도 4가지 색으로 색칠이 가능하다는 정리.
- 1976년에 Kenneth Appel과 Wolfgang Haken이 ‘증명’.
  - 핵심: 무한한 지도를 색칠하는 게 아닌 유한한 지도를 색칠하는 게 충분하다는 걸 증명.
  - 대표 지도: 1,482개의 지도를 찾아서 컴퓨터로 >1000 시간을 걸려서 색칠.
  - 대표 지도를 찾는 과정은 400페이지 → 맞는지 확인하기 엄청 힘듦.
  - 대표 지도를 색칠하는 건 컴퓨터 알고리즘 → 사람이 쓴 코드를 믿어야함.

Four Color Theorem (1/2)

수학에서 Proof Assistant의 사용

- 한계적인 발전: 신뢰할 수 없는 프로그램을 쓴다는건 변하지 않음.
- Coq을 통한 증명: 2005년에 Benjamin Werner와 Georges Gonthier가 증명.
  - 633개의 대표 집합을 찾는 과정: 찾은 집합을 색칠하는 게 충분하다는걸 증명.
  - 633개의 집합을 색칠하는 과정: 색칠이 옳다는걸 증명.
  - 증명 소스코드: https://github.com/coq-community/fourcolor
- 400페이지의 귀찮은 수학 + 컴퓨터 프로그램 신뢰 → Coq 구현 신뢰.



Feit Thompson Theorem

수학에서 Proof Assistant의 사용

- “Every finite group of odd order is solvable”.
- 1962, 63년에 250 페이지 이상의 증명.
- 2012년에 6년간 노력 끝에 Coq으로 증명이 끝났음을 알림.
  - 150,000 줄의 proof script.
  - 최종 결과물.
- 수학계에선 그다지 큰 관심을 받지 못함.
- Q: 과연 proof assistant가 현대수학을 할수 있을까?

```
Qed.

Theorem Feit_Thompson (gT : finGroupType) (G : {group gT}) :
  odd #|G| -> solvable G.
Proof. exact: (minSimpleOdd_ind no_minSimple_odd_group). Qed.

Theorem simple_odd_group_prime (gT : finGroupType) (G : {group gT}
  odd #|G| -> simple G -> prime #|G|.
Proof. exact: (minSimpleOdd_prime no_minSimple_odd_group). Qed.
```

증명 리뷰를 혼자서 하기 (1/2)

수학에서 Proof Assistant의 사용

- December 2021: Thomas Bloom은 다음을 증명.
- “Any set $A \subset \mathbb{N}$ of positive upper density contains a finite $S \subset A$ such that $\sum_{n \in S} \frac{1}{n} = 1$”.
- 6개월뒤 Bhavik Mehta와 Coq으로 증명.
  - 유용한 테크닉의 증명도 포함함.
- 이 논문은 아직 리뷰를 받지 않았음.

증명 리뷰를 혼자서 하기 (2/2)

수학에서 Proof Assistant의 사용

- 이 일이 왜 중요한가?
  - 수학자들이 Proof Assistant에 관심이 없던 이유 중 하나는 현대수학을 할 준비가 X.
  - Proof Assistant가 현대수학을 증명할 수 있음을 보여줌.
- (희망사항) 리뷰가 단순해질 수 있음.
  - 수학 논문은 증명 검증 때문에 리뷰가 매~~우 오래 걸림 (1~2년도 가능).
  - Proof Assistant가 널리 사용되면 증명 검증에 걸리는 시간이 줄어들 수 있을 것.

전산학에서 Proof Assistant의 사용

프로그램이란 무엇인가?

전산학에서 Proof Assistant 사용

- 알고리즘이란 무엇인가: 어떠한 수학적인 문제를 푸는 기계적인 과정.
- 프로그램은 알고리즘의 구현체.
  - 프로그램을 올바르게 작성했다는걸 어떻게 확신하는가?
  - 프로그램이 작성된 언어가 CPU에서 올바르게 실행된다는 걸 어떻게 확신하는가?
- Proof Assistant의 목적: 수학을 잘 쓰고 싶다.
  - 수학으로 올바름을 확신하고 싶다.
  - 수학을 올바르게 썼다는 것도 확신하고 싶다.

프로그램을 증명하는 방법

전산학에서 Proof Assistant 사용

- 명세: 프로그램에 원하는 성질의 수학적 표현.
- 구현: 위 성질을 만족하는 알고리즘의 구현.
- 증명: 구현한 프로그램이 실제로 명세를 만족한다는 걸 보이는 것.
- 중요성: 전산의 입장에서는 ‘어떻게 증명했냐’ 보단 ‘증명한게 유용하냐’가 훨씬 중요.
  - ‘증명’의 신뢰는 Proof Assistant로 넘김.
  - 증명한 ‘명세’의 유용성만 열심히 설명하면 됨.
- 얼마나 어려운가 = 얼마나 노오오동을 해야 하나.
  - 간단한 알고리즘: array에 원소 두개를 swap. (4 대 12)
  - 어려운 알고리즘: use-after-free/ABA를 방지하는 SMR 알고리즘. (140 대 2300++)

언어란 무엇인가?

전산학에서 Proof Assistant 사용

- CS101에서 다들 Python으로 코딩을 해봤음.
- 자신이 쓴 Python 코드가 엄밀하게 어떠한 방식으로 작동하는지 서술할수 있을까?
  - 예: x = 1; y = x + 1가 있을때, 왜 y값은 2가 되는가?
- 필요성: 엄밀해야 언어를 해석하는 도구 만들기가 가능함.
  - assembly로 변환하는 컴파일러 (C/C++/Java/Rust)
  - 해석하는 인터프리터 (Python/Java/JavaScript)
- 자세한 내용은 CS320 Programming Languages에서

C: 모든 언어의 기본

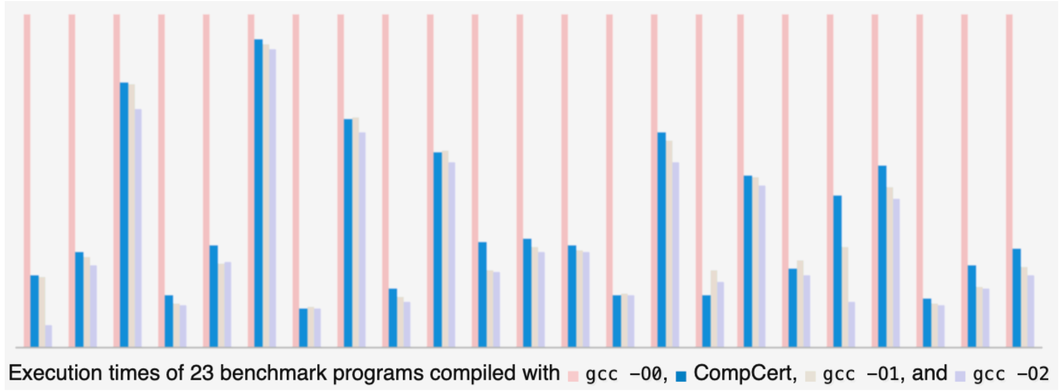
전산학에서 Proof Assistant 사용

- 처음으로 만들어진 “현대적인 프로그래밍 언어”.
- 모든 OS/효율적인/사이즈가 작은/다른 언어의 컴파일러 프로그램에 필수적으로 사용.
  - 안전하게 작동해야 하는 이유가 매우 있음.
  - 써보면 알겠지만 올바르게 쓰기 매우 힘듦.
- 엄밀한 언어의 정의가 존재.
  - Q: 그럼 뭐가 문제죠?
  - A: 701페이지입니다.
- 올바른 컴파일러를 만들기가 매우 힘듦.

CompCert: 검증된 C 컴파일러

전산학에서 Proof Assistant 사용

- (거의 대부분의) 구현이 검증된 C 컴파일러.
  - IR 변환의 증명.
  - 최적화의 증명.
  - 생성된 assembly가 C 코드와 ‘동일하게 동작’.
- 상업적으로 경쟁성이 있는 C 컴파일러.
  - 프로그램 검증이 단순히 심심풀이가 아님을 보여줌.
  - 안정성이 매우 중요한 사업 (항공, 원자로 사업 등)에서 사용중.



동시성 프로그래밍 (1/4)

전산학에서 Proof Assistant 사용

- 프로그램에 주어진 일을 하나의 CPU에서 하는 것이 아닌 여러 CPU에 동시에 시키는것.
  - 예: 과제를 각자 푸는게 아닌 스터디를 만들어서 문제를 분배하기
- 현대 프로그램의 속도를 위한 핵심 요소.
  - GPU: 게임, 인공지능, 영상매체.
  - NPU/TPU: 특화된 인공지능 기능을 얼마나 동시성을 잘 살릴 것인가?
- 저희 교수님: 앞으로는 700년은 동시성 프로그램이 주가 될 것.

동시성 프로그래밍 (2/4)

전산학에서 Proof Assistant 사용

- 이상: 알아서 깔끔하게 일이 분배됨.
- 

- 현실:
- 

- 동시성 프로그램의 올바름은 따지기가 매우 어려움.
- 왜 그런지 더 궁금하다면 내년에 CS431에서 ~~고통을~~ 공부를.

동시성 프로그래밍 (3/4)

전산학에서 Proof Assistant 사용

- 동시성 프로그램의 명세: 올바른게 뭔지 따지는 게 어려움.
  - 예: 학생한테 과제를 시켰는데 두명이 반씩 섞어서 제출한 후 두명 다 했다고 주장함.
- 동시성 프로그램의 구현: 올바른게 구현하는게 뭔지 따지는 게 어려움.
  - 컴파일러 최적화, 다른 함수와의 연관, OS 기능 등…
- 동시성 프로그램의 증명: 구현이 올바름을 어떻게 만족하는게 따지는 건 진짜 어려움.
  - “동시성이 있는 언어란 무엇인가”를 따지는 게 굉장히 힘들기 때문.
- Proof Assistant을 사용해서 증명한 결과물이 옳다는걸 남들도 납득시키는게 가능함.

동시성 프로그래밍 (4/4)

전산학에서 Proof Assistant 사용

- 컴파일러 최적화: 프로그램의 순서를 변경해서 더 효율적인 프로그램으로 변경.
- Speculative write: 나중에 있는 write instruction을 위로 이동시키기.
  - r := y; → x:= 1;
  - x := 1; → r:= y;
- (1) 이러한 최적화가 가능한 언어를 (2) 검증하기 위한 논리 체계를 만들기.

마치며

증명을 코딩하기

- 증명을 믿는 것은 그 증명을 한 사람을 믿는 것.
- Proof Assistant는 그 신뢰를 임의의 사람이 아닌 하나의 소프트웨어로 바꿈.
- 수학에선 난이도가 높은 증명을 검증하거나 알고리즘적인 증명을 확인 가능.
- 전산학에서는 수학을 올바르게 썼다는 확신을 안겨줌.
- 수학에서 어떻게 사용하는 지가 더 자세히 궁금하면 Lean을 알아보는걸 추천.
- 전산에서 어떻게 사용하는 지가 더 자세히 궁금하면 저희 연구실로 오세요.
  - https://cp.kaist.ac.kr/join
- 슬라이드 및 코드: https://github.com/Lee-Janggun/2022-09-26-M_square

References

- Kevin Buzzard: The rise of formalism in mathematics
- CompCert
- The Four Color Theorem
- On a density conjecture about unit fractions
- A Machine-Checked Proof of the Odd Order Theorem