Program Reasoning

1. Introduction

Kihong Heo

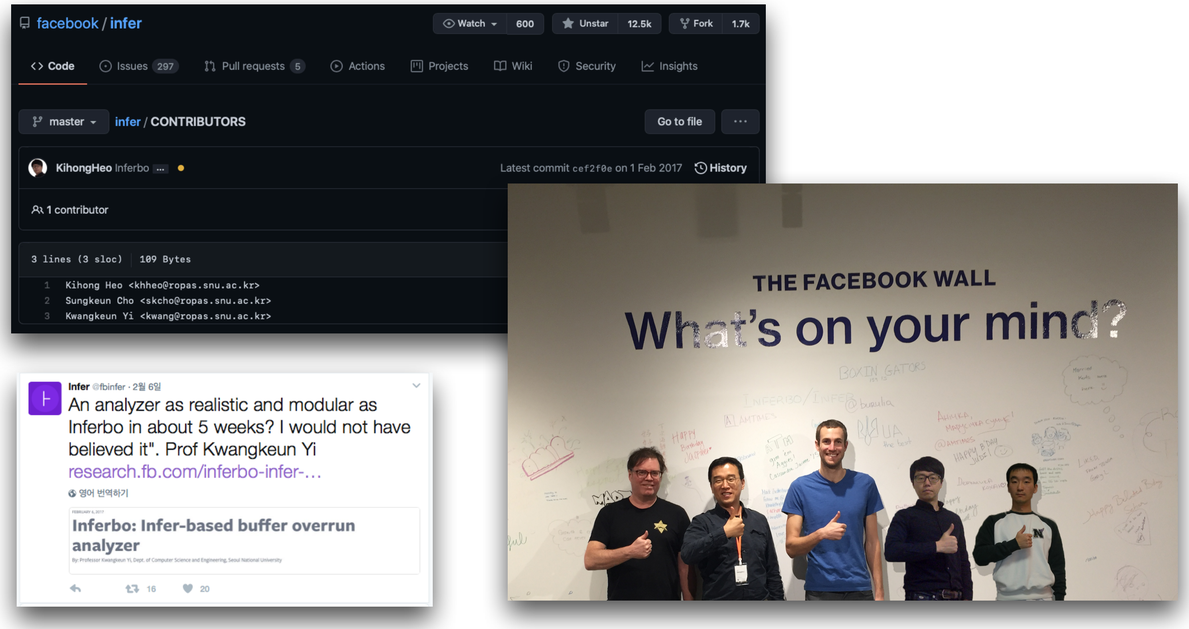
KAIST

About Me

- Instructor: Kihong Heo (허기홍, kihong.heo@kaist.ac.kr)
- KAIST CS / GSIS / Programming Systems Lab.
- Homepage: https://kihongheo.kaist.ac.kr / https://prosys.kaist.ac.kr
- Office: N5 2321
- Office Hours: after each class (by appointment)

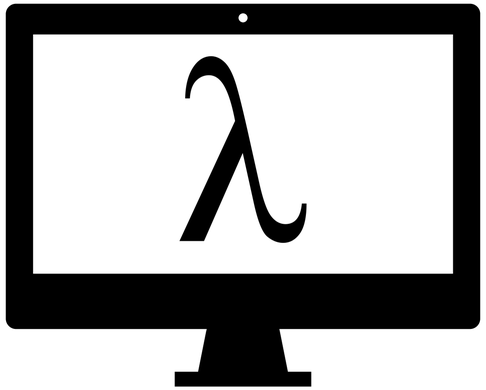
My Research

- Goal: solid PL theories ⟺ powerful programming systems
- Keywords: programming language, program analysis, SW engineering, SW security
- Good memories:



*https://research.fb.com/blog/2017/02/inferbo-infer-based-buffer-overrun-analyzer/

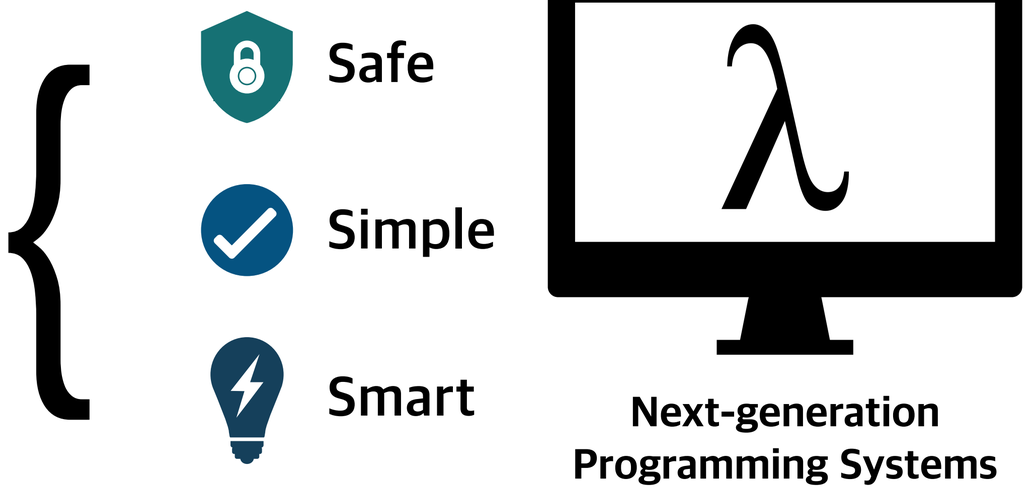
My Research



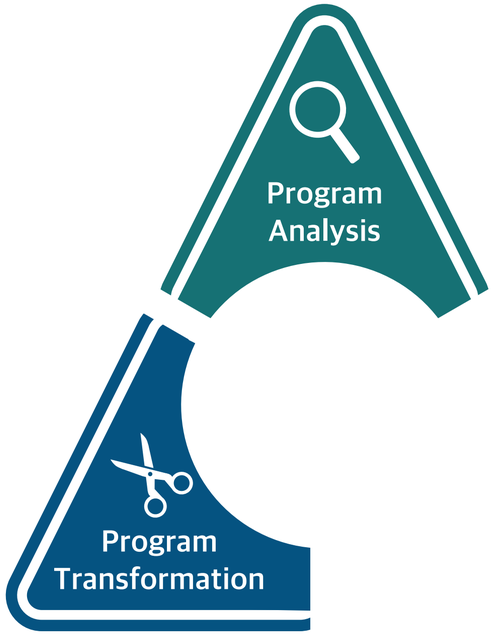
Next-generation
Programming Systems

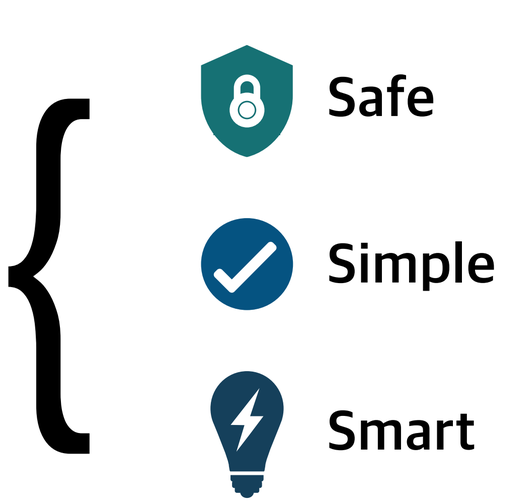
My Research

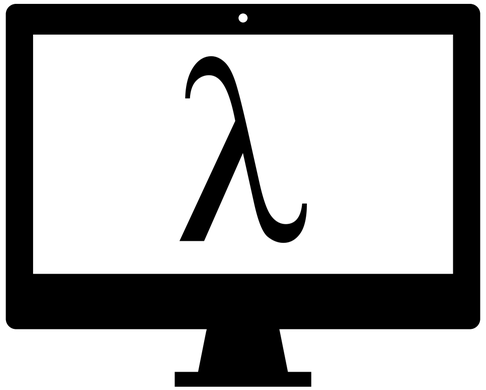

Program
Analysis


Safe
Simple
Smart
Next-generation
Programming Systems

My Research

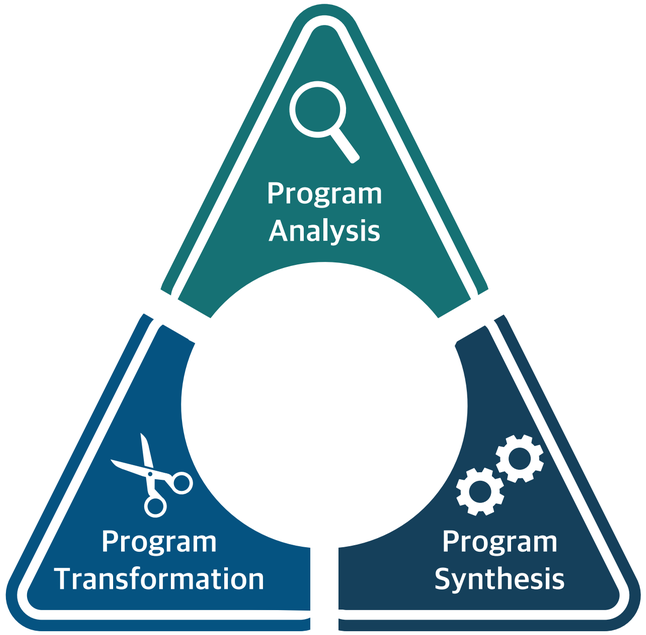






Next-generation Programming Systems

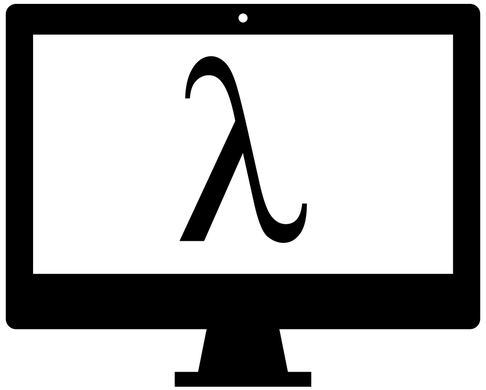
My Research



Safe

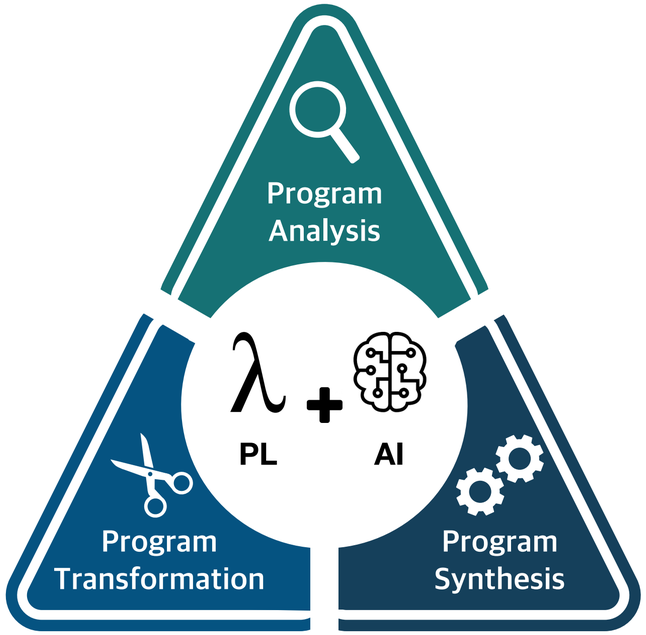
Simple

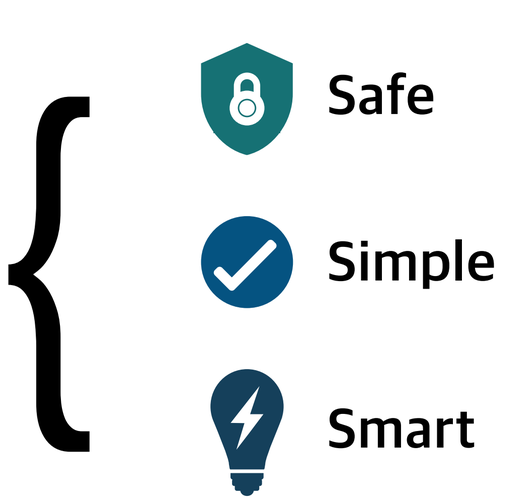
Smart

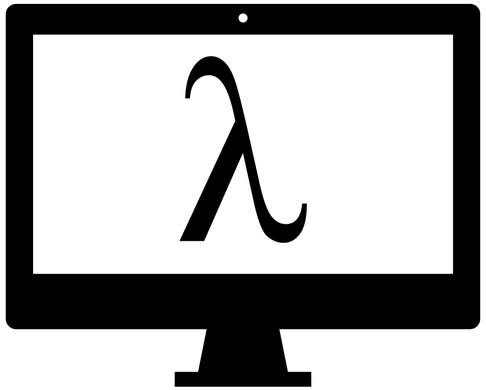


Next-generation Programming Systems

My Research







Next-generation Programming Systems

Course Information

- Course Website: https://github.com/prosyslab-classroom/cs492-program-reasoning
- Q&A Board: https://github.com/prosyslab-classroom/cs492-program-reasoning/issues
- TAs (mailing list: cs492.ta@prosys.kr)
  - Seungwan Kwon (권승완, seungwan.kwon@kaist.ac.kr)
  - Sujin Jang (장수진, sujin.jang@kaist.ac.kr)
- Textbook:
  - Lecture slides will be provided
  - See the course webpage

Grading

- Homework: 50%
- Final exam: 40%
- Participation: 10%
  - Active participation including questions or discussions (online or offline)
- NOTE: nonnegotiable!
  - DO NOT send an email for a negotiation (cheating)

Important Notice (1): Academic Integrity

- DO NOT share the course contents (e.g., assignments or exams) with others
  - Esp., Github public repository, chegg.com, etc
- DO NOT discuss the details of solutions with others
- DO NOT plagiarize
  - We keep all the submissions from previous years!
- Any integrity violation: at LEAST F
- If you have questions: QnA board > TAs > instructor

전산학부 명예규정 2022 봄학기 / School of Computing Honor Code 2022 Spring

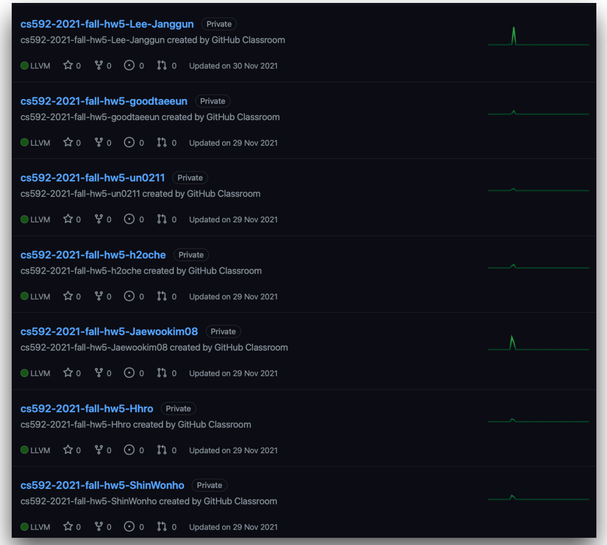
카이스트 전산학부가 운영하는 모든 수업에 참여하는 학생은 개인의 명예와 타인의 권리를 함께 존중하며 성실성과 정직성을 지키기 위하여 최선을 다합니다. 모든 시험 및 과제물 작성에 있어 허가되지 않은 어떤 형태의 도움도 받지 않습니다. 다음의 행위들은 학업의 성실성과 정직성을 위반하는 것으로 간주됩니다:

- 본인 이외의 사람/기관이 작성한 답안지, 숙제, 프로그램 소스 코드, 보고서 등을 참고 및 이용하는 행위
- 시험 및 과제물과 관련해 chegg.com과 같이 정답을 공유하는 온라인 서비스를 이용하는 행위
- GitHub 등의 코드 저장소에 본인의 과제 답안을 공개하거나, 타인이 공개된 답안을 참고 및 이용하는 행위
- 다른 학생이 본인이 작성한 답안지, 숙제, 프로그램 소스 코드, 보고서 등을 참고하도록 용인하는 행위
- 다른 학생이 작성한 결과물을 자신의 것인 양 제출하는 행위
- 다른 학생을 대신해 시험을 치루는 행위
- 개인이 수행하도록 되어있는 take-home 시험이나 과제물 작성에 있어 허락 없이 공동 작업을 하거나 부적절한 도움을 받는 행위
- 표절: 깊이와 무관하게 적절한 인용이나 언급 없이 타인의 창작물(참고서적, 문헌, 온라인상의 자료)을 무단으로 사용하는 행위

규정 위반 여부의 판단과 처벌 수위는 교과목 담당 교수에 의해 결정됩니다. 모든 부정행위는 전산학부 학사주임 교수 및 학부장님께 보고되며, 적발시 전산학부 내부적으로 아래와 같은 제약을 받습니다.

- 향후 2학기 동안 모든 포상 및 장학금 수여/추천 대상에서 제외함
- (주전공이 전산학부가 아닌 경우) 향후 2학기 동안 전산학부로의 전과 금지

위반의 심각성에 따라 학교 전체의 상벌위원회에 회부될 수 있습니다. 카이스트 학사규정이 허용하는 징계의 범위는 아래 첨부된 학생 징계 양형 기준을 참고하세요 (학생 핸드북 한글판 67페이지 , 다운로드는 https://portal.kaist.ac.kr/ennotice/student_notice/11614934649338)



cs592-2021-fall-hw5-Lee-Janggun Private
cs592-2021-fall-hw5-Lee-Janggun created by GitHub Classroom
LLVM ☆ 0 ⑂ 0 ⊙ 0 ⇄ 0 Updated on 30 Nov 2021

cs592-2021-fall-hw5-goodtaeeun Private
cs592-2021-fall-hw5-goodtaeeun created by GitHub Classroom
LLVM ☆ 0 ⑂ 0 ⊙ 0 ⇄ 0 Updated on 29 Nov 2021

cs592-2021-fall-hw5-un0211 Private
cs592-2021-fall-hw5-un0211 created by GitHub Classroom
LLVM ☆ 0 ⑂ 0 ⊙ 0 ⇄ 0 Updated on 29 Nov 2021

cs592-2021-fall-hw5-h2oche Private
cs592-2021-fall-hw5-h2oche created by GitHub Classroom
LLVM ☆ 0 ⑂ 0 ⊙ 0 ⇄ 0 Updated on 29 Nov 2021

cs592-2021-fall-hw5-Jaewookim08 Private
cs592-2021-fall-hw5-Jaewookim08 created by GitHub Classroom
LLVM ☆ 0 ⑂ 0 ⊙ 0 ⇄ 0 Updated on 29 Nov 2021

cs592-2021-fall-hw5-Hhro Private
cs592-2021-fall-hw5-Hhro created by GitHub Classroom
LLVM ☆ 0 ⑂ 0 ⊙ 0 ⇄ 0 Updated on 29 Nov 2021

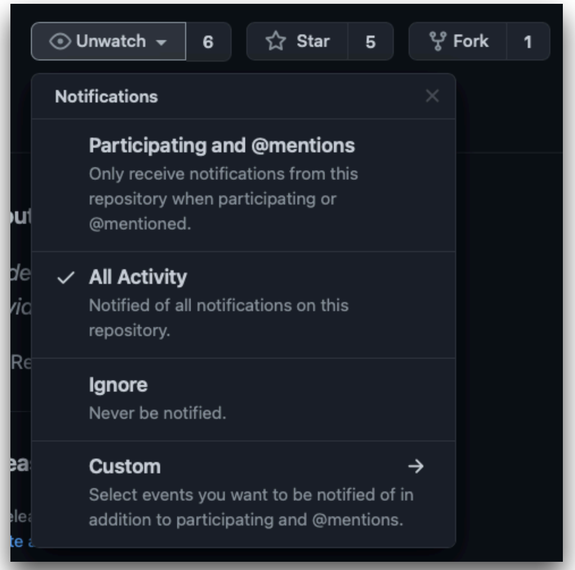
cs592-2021-fall-hw5-ShinWonho Private
cs592-2021-fall-hw5-ShinWonho created by GitHub Classroom
LLVM ☆ 0 ⑂ 0 ⊙ 0 ⇄ 0 Updated on 29 Nov 2021

Important Notice (2): In-class Operations

- Language: English (default), Korean (supplementary)
- Attendance: always (default), absence (if necessary)
  - No quantified attendance score
  - “What is essential is invisible to the eye” - The Little Prince
  - I expect you to be here, as you expect me to be here!
- Questions & discussion (either in Korean or in English): highly encouraged

Important Notice (2): Out-of-class

- All Q&A and public notices: Github issue board
  - “Watch” all notifications
- Private notices (grading, etc): KLMS
- Questions are always welcome except for
  - Too detailed ones (TAs are not debuggers!)
  - Directly related to the solutions
- Actively discuss with your classmates



A long time ago
in a galaxy far, far away….

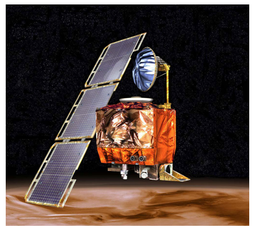
SOFTWARE BUGS

Software Bugs: A Persistent Problem

- A long time ago, far far away

The Patriot Missile (1991)
Floating-point roundoff
28 soldiers died

The Ariane-5 Rocket (1996)
Integer Overflow
$100M



NASA's Mars Climate Orbiter (1999)
Meters-Inches Miscalculation
$125M

- Unfortunately, it becomes your own problem now

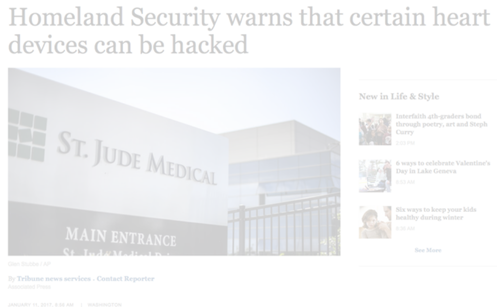
Software Bugs: A Persistent Problem

- A long time ago, far far away
- U

COST OF A SOFTWARE BUG

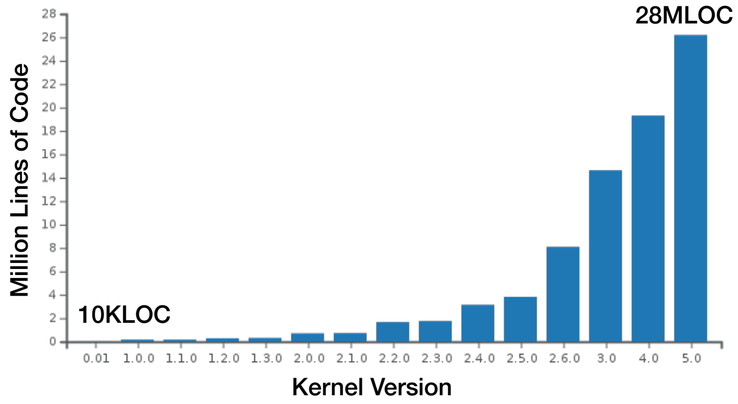
$100	$1,500	$10,000
If found in Gathering Requirements phase	If found in QA testing phase	If found in Production

- IBM Systems Sciences Institute, 2015

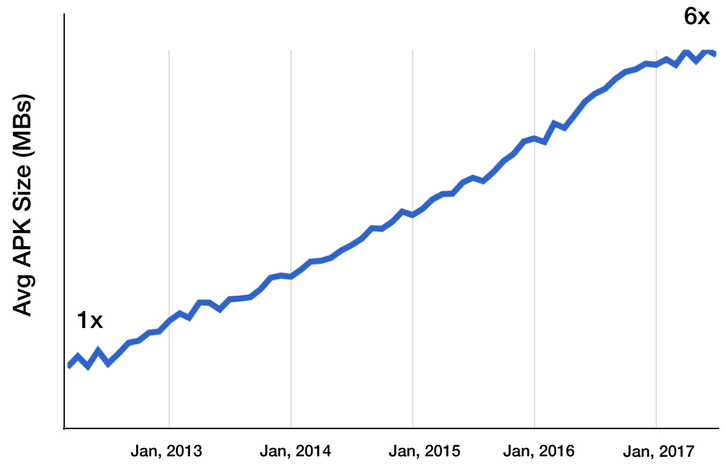


Why Software Still Fails?

Size of Linux Kernel



Avg. Size of Android Apps



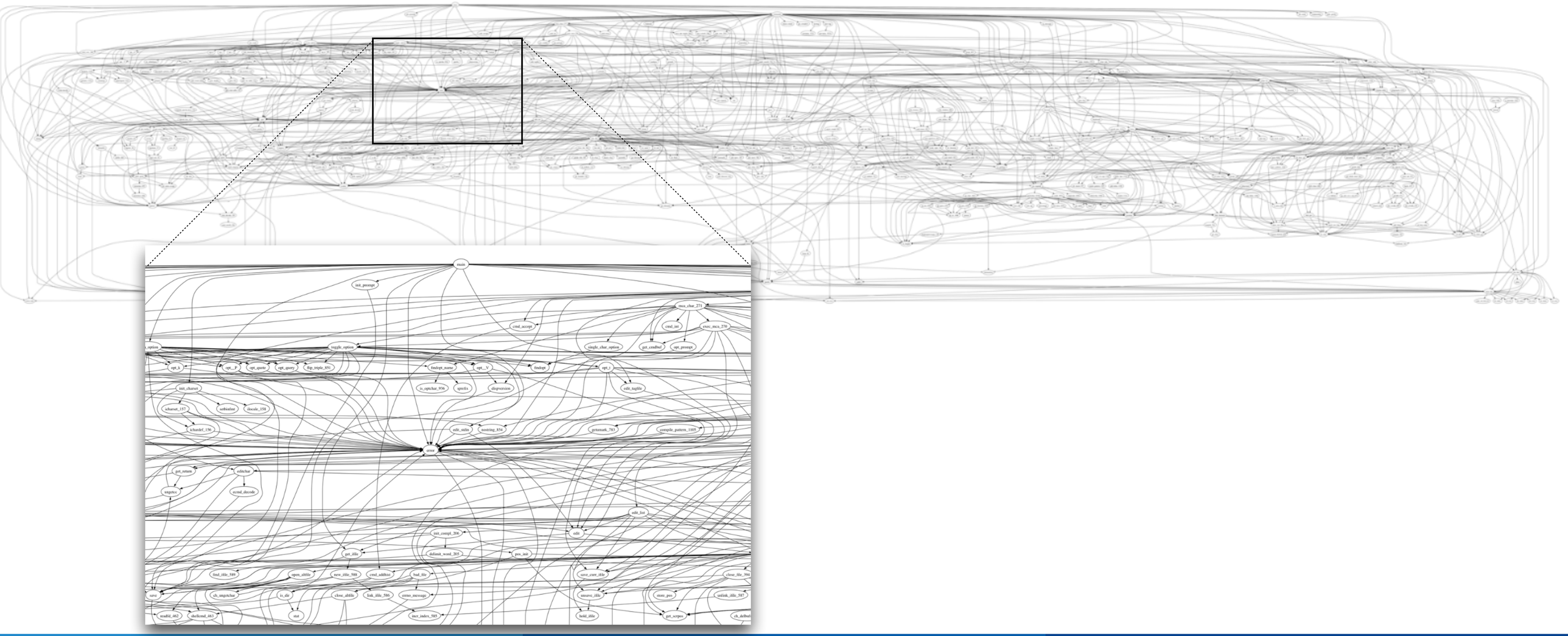
X

GitHub

10M+ New Developers
44M+ New Repositories
87M+ New Pull Requests
in 2019

Software Complexity

less-382 (23,822 LOC)



What is the Future of Programming?

How to automatically **prove** if a given program satisfies the specification?

How to automatically **generate** a program that satisfies the specification?

Course Objectives: Principles

- Program verification
  - How to specify specification?
  - How to generate a correctness proof?
- Program synthesis
  - How to specify specification?
  - How to generate a correct program?

Success Stories

Program Verification

Windows Device Driver

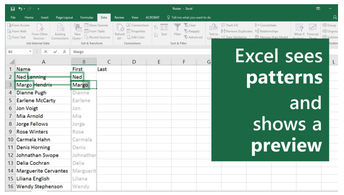
Microsoft

Astrée

Airbus Controller

ENS / AbsInt

Program Synthesis



Excel FlashFill

Microsoft

Copilot

Github

Course Objectives: Opportunities

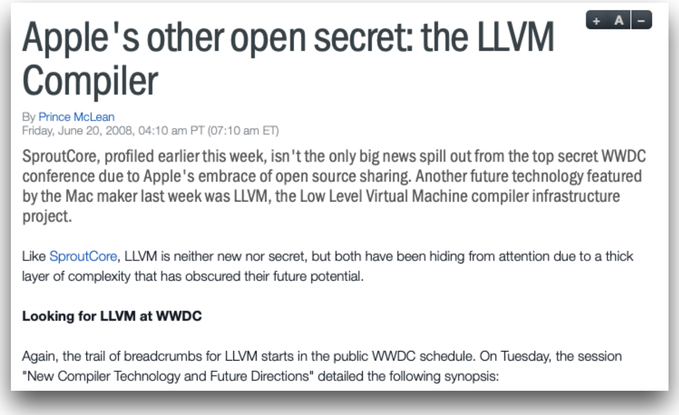
- What is the state-of-the-art? What are the applications?
  - Program verification: API protocol, compiler, web browser, OS, DNN, …
  - Program synthesis: cryptography, de-obfuscation, programming-by-example, …
- What is your own application?

Course Objectives: Practice & Challenge

- Homework: **design** & **implement** program analyzers
  - 6 (main) + 3 (dummy) assignments
- Programming assignments in OCaml using LLVM & Z3
  - You will write your verifier & synthesizer in OCaml
  - Your verifier will analyze LLVM IR code
  - You will utilize Z3 in your verifier & synthesizer
- Why LLVM? (https://llvm.org)
- Why OCaml? (https://ocaml.org)
- Why Z3? (https://github.com/Z3Prover/z3)

The LLVM Compiler Infrastructure

- The de-facto standard & well-structured compiler toolchain
  - parser, code optimizer, linker, loader, debugger, etc
- A wide variety of frontends: C/C++, Obj-C, Swift, Fortran, etc
  - translated to the LLVM IR (intermediate representation)


Apple's other open secret: the LLVM Compiler

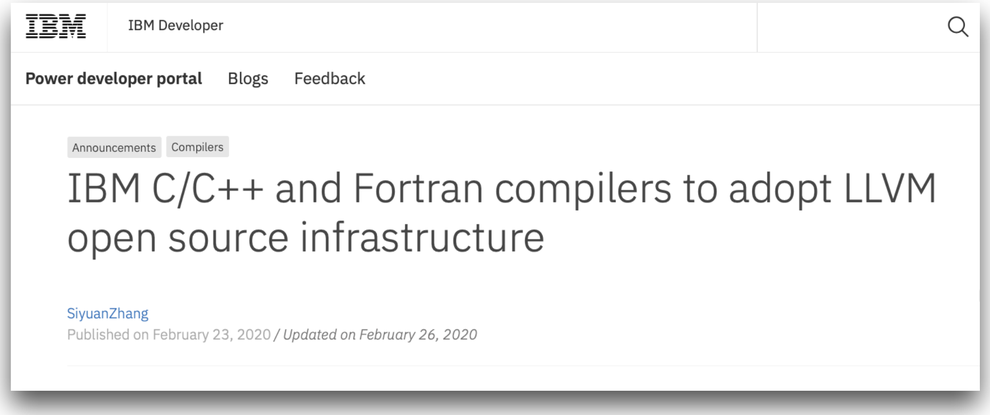
By Prince McLean
Friday, June 20, 2008, 04:10 am PT (07:10 am ET)

SproutCore, profiled earlier this week, isn't the only big news spill out from the top secret WWDC conference due to Apple's embrace of open source sharing. Another future technology featured by the Mac maker last week was LLVM, the Low Level Virtual Machine compiler infrastructure project.

Like SproutCore, LLVM is neither new nor secret, but both have been hiding from attention due to a thick layer of complexity that has obscured their future potential.

Looking for LLVM at WWDC

Again, the trail of breadcrumbs for LLVM starts in the public WWDC schedule. On Tuesday, the session "New Compiler Technology and Future Directions" detailed the following synopsis:



The OCaml Language

- Simple, safe, and realistic programming language
  - Strong type system, higher-order functions, etc
- Official OCaml bindings to LLVM and Z3 API supported
- A lot of growing demands from academia and industry
- See the materials on the course webpage

The Z3 Theorem Prover

- State-of-the-art automated theorem prover by Microsoft Research
- Solving satisfiability modulo theory (SMT) problems
  - first-order logic with background theories
    (e.g., arithmetic, bit-vectors, arrays, datatypes, uninterpreted functions, etc)

Boolean Satisfiability Problem (SAT)

$$(\neg A \vee B) \wedge (\neg B \vee C) \wedge (A \vee \neg C \vee B)$$

Satisfiable when
A = false
B = true
C = true

Satisfiability modulo theory (SMT)

$$x + 2 = y \implies f(read(write(a, x, 3), y - 2)) = f(y - x + 1)$$

Arithmetic | Array | Uninterpreted Functions

Homework

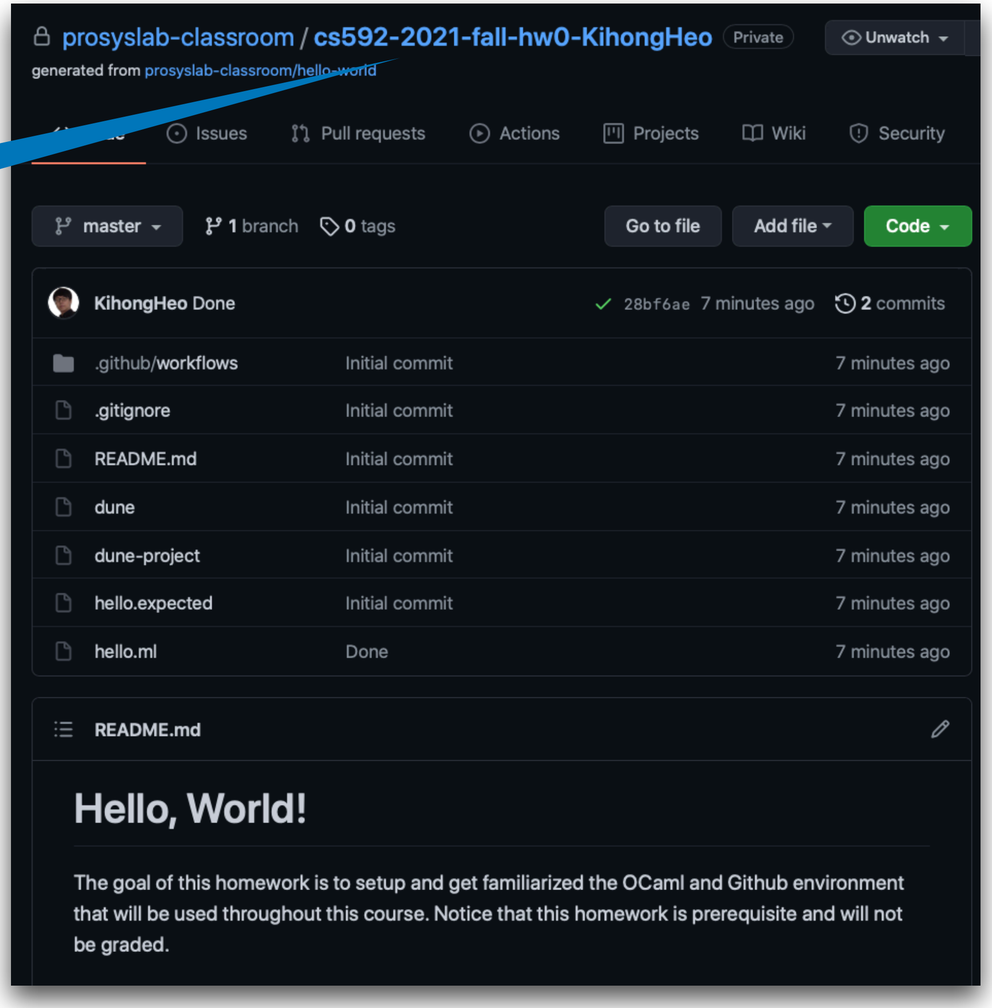
- All submissions will be managed using Github / Github Classroom
    1. For each HW, a unique invitation URL will be posted on the issue board
    2. Once you accept, a private repo for your assignment will be created
    3. You can push as many commits as you want before the deadline
    4. The final commit of your master branch will be graded
- Gradescope will be used for some assignments and exams
- 80% credit for 1-day late, 50% credit for 2-days late, NO credit otherwise

Homework 0.1: Hello, World!

- Goal: setting up and getting familiarized with OCaml and Git environments
  - Implement your “hello-world” program in OCaml
  - Test on your machine
  - Push to your Github repository
  - See the result in Github Action
- The invitation URL will be posted on the course webpage
- Will not be graded

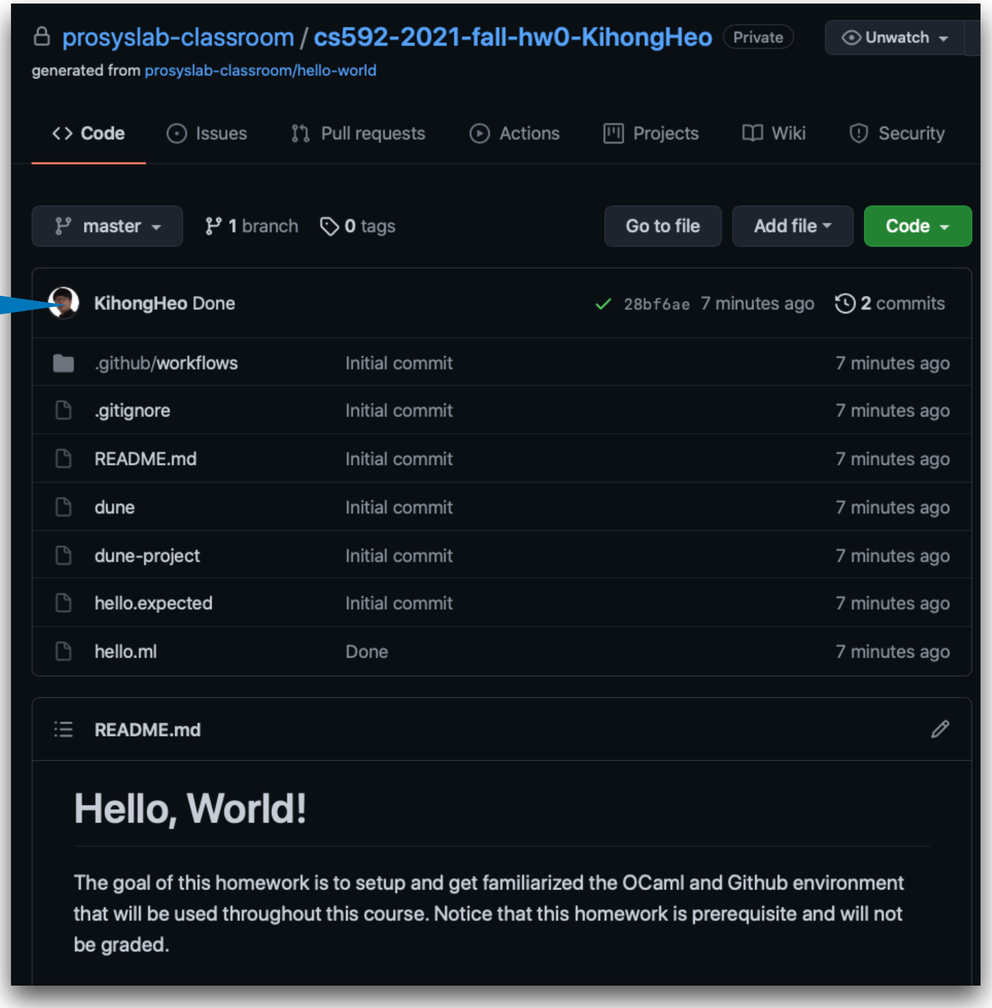
Homework 0.1: Hello, World!

1. Accept the invitation and have your repository



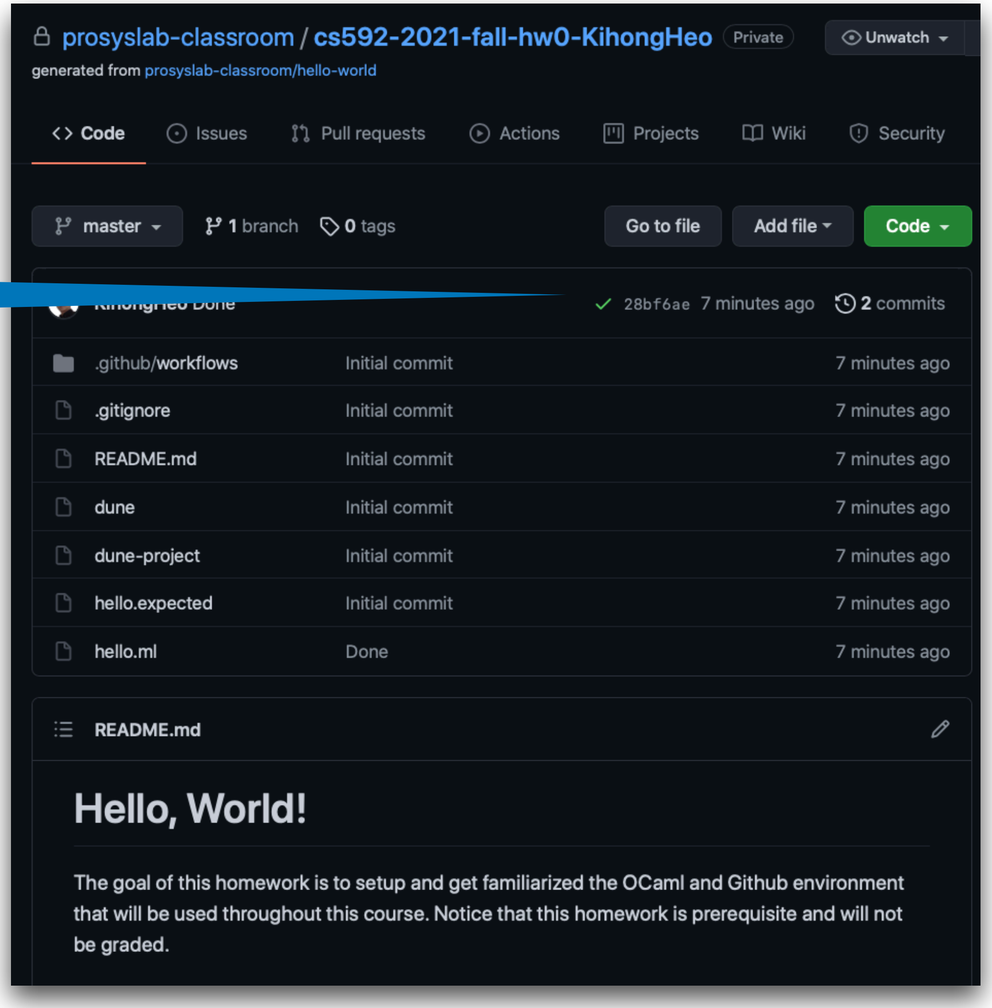
Homework 0.1: Hello, World!

2. Commit your code



Homework 0.1: Hello, World!

3. See your result



Homework 0.2: OCaml Programming

- Goal: getting familiarized with basic OCaml programming
  - Solve a few programming problems in OCaml
  - Test on your machine
  - Push to your Github repository
  - See the result in Github Action
- The invitation URL will be posted on the course webpage
- Will not be graded but highly recommended if you are not familiar with OCaml

Homework 0.3: Watching the Video

- Watch the Youtube video about “Undecidability” vis the course webpage
- Think about the followings:
  - What is the fundamental limitation of SW?
  - Why is it difficult to prove the correctness of SW?
  - How can we overcome the above challenges?
- Provide your opinion and write an essay
  - Not mandatory but a reward might be given depending on the quality
  - E.g., participation point, questions for the final exam, drink, fame, etc

Writing

- Syntactic requirements
  - Typeset your document in Latex using a provided template (maximum 2 pages)
  - Korean (if you are a native Korean speaker), English (otherwise)
- Semantic requirements: Top-down (두괄식)
  - For each paragraph, write the topic sentence first followed by the details.
- See the README for the details
- The invitation URL will be posted on the course webpage

Misc

- Submit your Github account via the google form (see the Github issue board)
- Join Gradescope (see the Github issue board)
  - For written assignments and the final exam
- Rules for programming assignments
  - Preserve the structures (directories, files, types, etc)
  - Don’t install further Github App