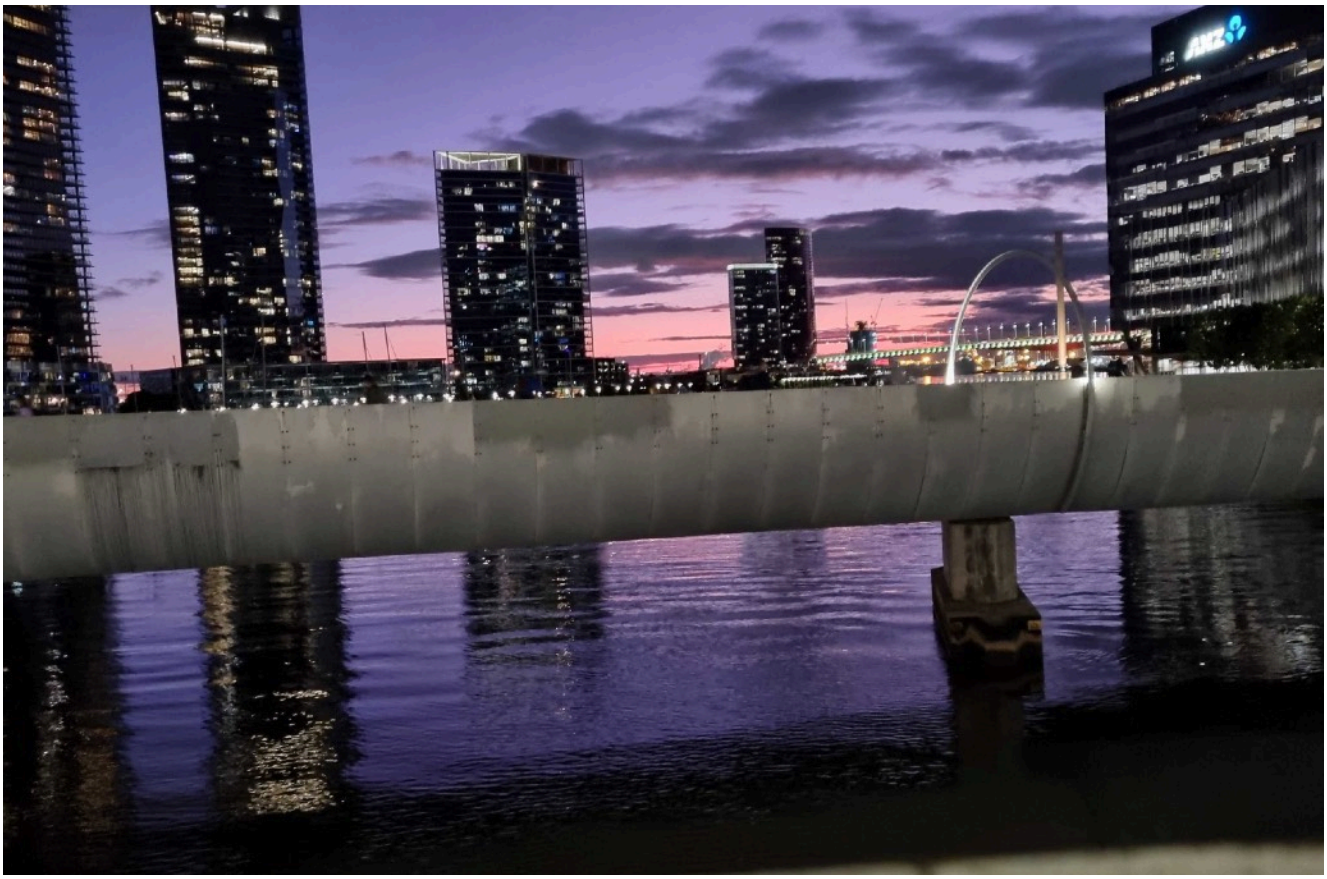


ICSE 2023 Trip Report

호주 멜버른

23. 05. 13. ~ 23. 05. 21

고려대학교 홍성준

소프트웨어공학 분야의 최고 학회인 ICSE에 다녀왔다. 호주는 어느 나라에서든 멀텐데, 가장 살기 좋은 도시라는 멜버른에서 열린 탓인지 사람들이 많이 왔다. 발표 도중에 출입문이 열리면 바깥에서 쉴새 없이 떠드는 소리에 발표가 방해될 정도로 사람이 많았다. 또 학회장에 직접 와서 보니 아시아 사람들, 특히 우리나라 사람들의 활약이 대단했고 개인적으로도 많은 자극이 되었다. 배우고 느낀점들이 잘 공유 됐으면 좋겠다.

APR과 Transformer 모델

이번 학회 참가의 주 목적은 GPT 광풍이 APR 분야에 끼친 영향을 직접 체감하기 위함이었다. ChatGPT가 공개되어 대중의 관심을 끌기 시작한건 작년 겨울쯤이지만, CS 분야에서는 이미 일찍이 트랜스포머 기반 인공 신경망의 탁월한 성능을 주목해왔다. 작년 FSE에서 조짐이 보이더니, 이번 ICSE에선 아예 *Program repair with and for AI* 세션이 생겼다. 이 세션에서는 주로 트랜스포머 기반 모델을 사용하는 APR 연구들이 발표되었고, 요즘 핫한 미리 학습된 LLM 모델의 APR 성능 연구에 관한 스터디도 두 개 발표됐다. 다른 APR 세션에서도, 버그 리포트를 활용하여 Fault Localization 성능을 개선하는 연구[1] 하나를 제외하면 모두 직/간접적으로라도 트랜스포머를 활용한 APR 연구가 발표됐다. 내가 본 것만 총 9편의 APR 논문 중에 8편이 트랜스포머를 활용한 연구였다. 이제는 딥 러닝에 관한 기본적인 지식 없이는 내 관심사인 APR 분야의 주요 흐름을 따라가지 못하는 지경에 이르렀다. 내 전공 분야가 좁혀지면서 CS 전반의 트렌드에 관심이 약해지고 잘 못따라가고 있다는 생각이 어렴풋이 들었었는데, 이번을 계기로 적어도 모두가 관심있어하는 기술정도는 틈틈히 공부해놔야겠다는 생각이 들었다.

트랜스포머의 기술적 파급력이 워낙 크고, APR 특성상 그 적용이 자연스럽기 때문인 것일까. 덩달아 APR 분야의 관심도도 증가한 것 같다. 특히 *Program repair with and for AI* 세션은 정말 발 디딜틈 없이 사람이 꽉차 있어서, 서서 듣는 것도 어려운 수준이었다. 체어도 신기했는지 본인 세션이 얼마나 활황인지 아예 셀카로 인증샷을 남길 정도였다. *APR* 성능 역시 트랜스포머의 힘을 빌려 꽤 많이 올라갔는데, 발표된 수치에 따르면 현재 최고 성능은 Correct Patch Rate 15.8% (62/393)다[2]. 근 5년간 약 12% 대에 수렴하던 수치라는 걸 감안하면, 1년 만에 비약적인 발전이다.

다만 “가장 높은 성능을 내면 장땡”이라는 분위기가 APR 연구에도 만연하는 것 같아 아쉽다. 주로 성능 평가에서 APR 연구자의 관점에서는 “잉?” 하는 부분들이 눈에 많이 띈다. 예를 들어, LLM 모델 기반의 APR이 이미 일찌감치 기존의 나열 기반의 APR 성능을 뛰어넘었다는 결론을 내린 스터디 논문[3]이 있었는데, 실험을 이미 오류 수정 위치가 알려진 PFL (Perfect Fault

[1] Better Automatic Program Repair by Using Bug Reports and Tests Together

[2] Tare: Type-Aware Neural Program Repair

[3] Automated Program Repair in the Era of Large Pre-trained Language Models

Localization) 세팅에서만 수행했다. FL 기술은 APR 성능에 매우 큰 영향을 끼치고, 많은 기술들이 낮은 FL 성능때문에 타협하는 부분들이 많다. 모든 기술의 FL 성능을 통일해서 비교하기가 현실적으로 어렵기 때문에 이러한 요소를 배제한 PFL이라는 보조 지표를 도입한 것인데, 이 연구에서는 스터디 논문임에도 불구하고 저 실험만 가지고 결론을 도출해서 황당했다. APR 고수인 Abhik Roychoudhury 교수님도 비슷한 질문을 했는데, 발표자는 동문서답을 길게했다.

흥미로운 발표

Better Automatic Program Repair by Using Bug Reports and Tests Together

Manish Motwani, Yuriy Brun

Fault Localization에서 버그 리포트와 테스트 실행 정보를 같이 쓰면 더 좋은 성능을 내고, 더 나아가 APR 에도 도움이 된다는 걸 보인 연구이다. FL 기술은 크게 테스트 실행 정보를 쓰는 SBFL (Spectrum-Based FL)과, 버그 리포트에 적힌 자연어 등의 정보를 활용하는 IRFL (Information-Retrieval-Based FL)로 나뉜다. 두 방식은 서로 보완적이라고 알려져있는데, 이 연구에서는 둘을 합친 SBIR 이라는 새로운 FL 기술을 제안한다. 다만 SBFL과 IRFL에서 새로운 기술적 기여가 있는 것은 아니고, 서로 다른 각 FL 기술의 랭킹을 적절히 aggregate 하는 것이 핵심이다. 이 과정에서 Rank Aggregation 알고리즘을 사용하는데, 이미 Information Retrieval, 선거 등의 분야에서 광범위하게 연구되어온 분야라고 한다.

Technical novelty를 잘 파악하기는 어려웠지만, 누구나 생각해볼만한 가설을 구체화 해서 하나의 연구로 마무리 했다는 점에서 좋았다. 특히 최근 FL 기술들이 APR 성능 관점에서 비교된 적은 없었기 때문에, 다른 APR 연구자들에게 많은 도움이 될 연구라 생각한다. 인력이 많이 들어갔을 법한 연구인데, 둘이서 어떻게 효율적으로 해냈는지 궁금하다. 특히 가장 좋은 성능을 얻기 위해 SBFL은 최신 GZoltar로 다시 돌리고, IRFL은 새롭게 구현까지 했다고 한다.

Concrat: An Automatic C-to-Rust Lock API Translator for Concurrent Programs

Jaemin Hong, Sukyoung Ryu

KAIST PLRG 홍재민 님의 발표로, 발표만 들어도 무엇을 어떻게 푼지 이해될 정도로 잘 했다. *Rust*는 올바른 *Lock API* 사용을 타입 시스템이 보장해주는데, 이를 활용하기 위해 *C* 프로그램의 *Lock API*를 *Rust*에 맞게 자동으로 변환해주는 문제를 풀었다. 이를 위해서 어떤 락이 어떤 데이터를 보호하는지, 어떤 프로그램 지점에서 얻어지는지를 정적 분석으로 알아낸다. 현재 *Concurrent C-to-Rust* 변환만을 위한 *Lock Relation* 분석은 없기 때문에, 문제가 유니크하고 그 자체로 해결법인 좋은 연구다. 어짜피 번역 후의 코드가 *Rust* 컴파일러의 타입 시스템을 통과해야 하므로, 컴파일러의 타입 시스템에 최적화된 정확도를 갖는 분석기를 디자인한 것이 핵심. 영어 실력도 출중하고 이번 *ICSE* 발표 중 가장 인상 깊었다.

Rete: Learning Namespace Representation for Program Repair

Nikhil Parasaram, Earl T. Barr, Sergey Mechtaev

이번 *ICSE*의 Distinguished Paper이다. 발표자인 Nikhil Parasaram은 알고보니 작년 FSE에서 알게 된 APR 논문 Trident의 1저자였다. 한번 더 알고보니 예전의 소순범 박사님과 스마트 컨트랙트 테스팅 연구를 할 때 알게된 정적 분석 도구 Mythril의 메인테이너였다. 어쩌다 APR로 넘어온지는 모르겠는데, 어쨋든 세상 참 좁다.

```
    if (
        is_decorator
        and self.previous_line
    ):
        return 0, 0
+ if (
+     is_decorator
+     and self.previous_line
+     and self.previous_line.is_comment
+ ):
+     return 0, 0

    newlines = 2
    if current_line.depth:
        newlines -= 1
```

(a) The developer fix for a bug in Black.

```
    if (
        is_decorator
        and self.previous_line
    ):
        return 0, 0

    if □[1] and □[2]:
        return 0, 0

    newlines = 2
    if current_line.depth:
        newlines -= 1
```

(b) A template for a hypothetical repair tool.

```
    if (
        is_decorator
        and self.previous_line
    ): # Code block used to generate fix
        return 0, 0

    if □[1] and self.previous_line.□[2]:
        return 0, 0

    newlines = 2
    if current_line.depth:
        newlines -= 1
```

(c) Edited locations with abstract variables.

Distinguished Paper라 기대했는데, 발표 듣고는 잘 파악이 안되서 논문을 찾아서 읽어봐도 정확히 뭘했는지 직관적으로 이해하기는 좀 어렵다. 제목의 "*Namespace*"는 헷갈리기만하고 큰 의미는 없고, 기본적으로는 빈 칸에 들어갈 적절한 프로그램 변수 (필드)가 무엇인지 추론하는 패

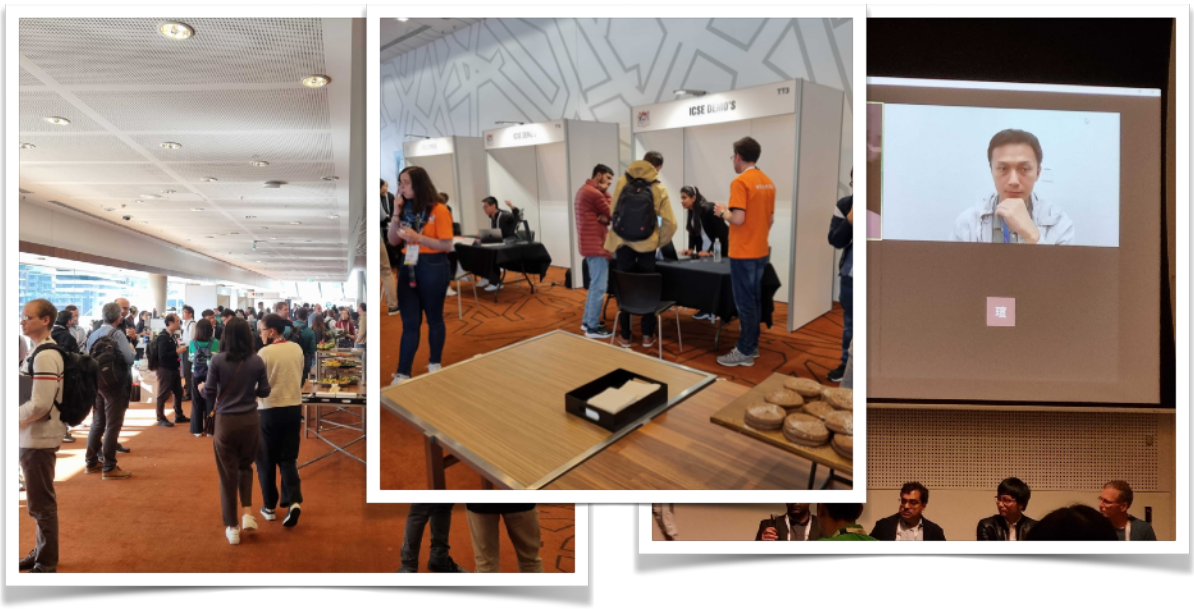
치 랭킹 문제를 푼 것이다. 변수 추론은 CodeBert를 미세 조정한 모델을 사용한다. 핵심은 수정 라인 주변 코드를 그대로 보는 대신, CDU (Conditional Def-Use) 라는 일종의 Def-Use 체인을 봄으로써, 훨씬 멀리 떨어져있는 변수의 의미 관계까지도 학습한다는 것이다. 결론적으로 위의 Figure (a)의 개발자 패치와 의미적으로 동일한 상당히 복잡한 패치를 1,000위 안에 생성한다고 한다.

	Patch Generator	
A	Angelix's Angelic Values	[38]
S	SOSRepair's Database of Snippets	[12]
P	SPR's Transformation Schemas	[14]
C	CoCoNut's Neural Machine Translation	[5]
E	Trident's Naïve template enumeration	[9]
G	GenProg's Genetic Algorithm	[39]
S	Plastic Surgery Algorithm	[8]
	Patch Prioritisation	
D	DirectFix's modification minimisation	[16]
C	CoCoNut's Neural Machine Translation	[5]
E	Trident's expression size minimisation	[9]
T	Trident's side effects minimisation	[9]
P	Prophet's Maximum Likelihood Estimation	[1]
S	Plastic Surgery Algorithm	[8]
	Variable Prioritisation	
E	Naïve variable enumeration	-
H	Heuristic discussed in Section V-B	-
B	CodeBERT	[31]
G	GraphCodeBERT	[40]
D	CodeBERT fine-tuned on DU chains	Section III-C
C	CodeBERT fine-tuned on CDU chains	Section III-C
F	Random Forest	Section V-B

제안한 기술 자체보다도 실험이 인상적이다. 이 연구에서 제안하는 *Variable Prioritization* 이라는 개념은, 임의의 패치 합성기에 독립적으로 적용 가능한 것은 물론, Prophet과 같은 기존의 패치 랭킹에까지 적용 가능한 상당히 일반적인 기술이다. 이를 보이기 위해 다양한 합성기를 고려하고, 상당히 다양한 APR Configuration 을 비교했다. 내가 지금 하고있는 연구에서 보이고자 하는 방향이 상당히 겹치는데, 좋은 참고 자료가 될 듯하다.

학회장

메인 컨퍼런스 전날 열린 *APR* 워크샵에 참가했다. 이번 워크샵이 *4*번째 개최인데 그동안 코로나로 원격으로만 열리다가 처음으로 현장에서 만난다고 했다. 참가자 대부분이 평소 논문에서 자주



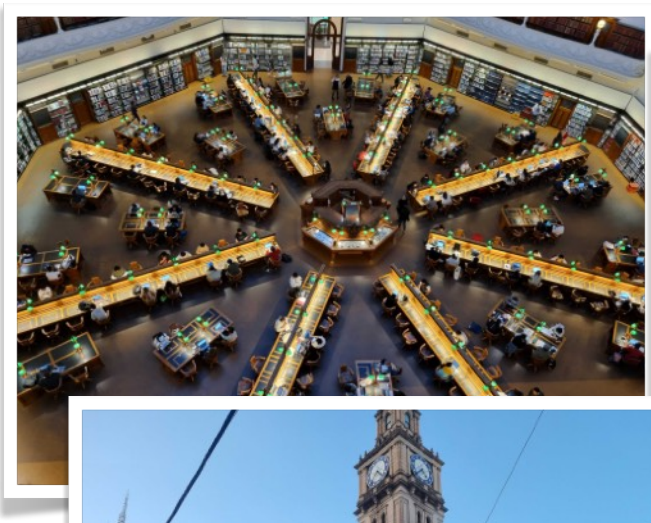
보던 사람들이었고, 비유하자면 SIGPL 계절 학교 느낌과 비슷했다. 다들 서로 아는 사이라 그런지 메인 컨퍼런스 세션보다는 다소 여유롭고 친숙한 분위기에서 진행됐다. NUS의 Abhik 그룹에서 SMT/SyGus-Competition과 비슷한 APR 경진 대회를 준비 중이라는 사실도 알게됐다. 논문 발표에 이어서 "*Challenges and Opportunities of APR*" 이라는 주제로 패널 디스커션이 진행되었는데, ChatGPT 시대에서 APR이 나아가야 할 방향을 토의하는 자리였다. 전체 내용을 다 이해하기는 어려웠지만, 전반적으로 커뮤니티에서는 좋은 기회라는 인식이 강한 것 같았다.

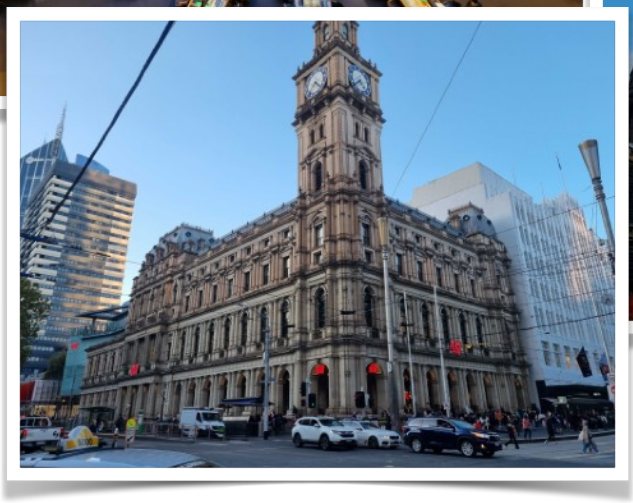
데모 세션이 발표 대신 부스 형태로 진행되었는데, 영어로 말걸기가 부담스러운 사람들은 데모 부스를 찾아가면 쉽게 얘기를 시작할 수 있다. 특히 둘째 날에는 NUS에서 Cerberus 라는 APR 프레임워크를 발표했는데, Prophet, Angelix, SAVER (!?) 등 여러 APR 기술을 한데 모아 쉽게 사용하고 통일된 환경에서 실행해주는 플랫폼이었다. 처음에 서로 잘 모르고 얘기를 하다가, SAVER 얘기를 하길래 내가 그 논문 저자라니까 서로 놀랐다. Docker 이미지가 없어서 플랫폼 위에서 세팅하는데 애를 먹었다고 한다. NPEX나 현재 하고있는 연구는 꼭 Cerberus 플랫폼에 직접 올려주겠다고 약속했다.

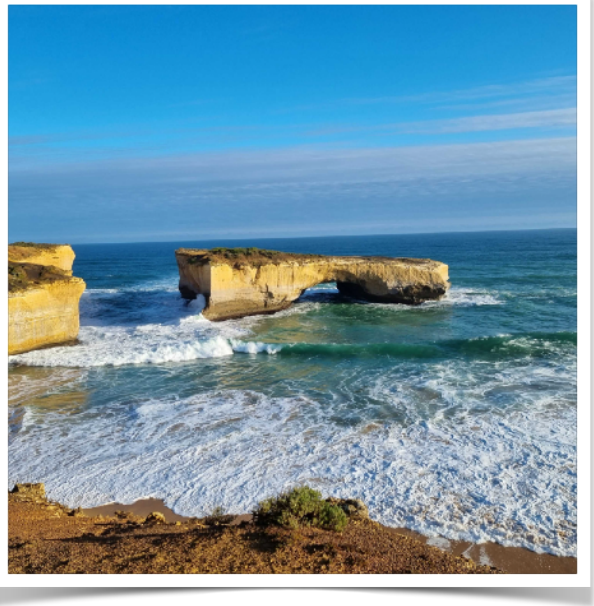
마지막 날엔 AlphaRepair 저자를 직접 만나 실험 재현에 관한 의문들에 대해 얘기를 나눴다. 재현이 안되고 논문과 실제 구현이 불일치하는 부분들에 대해 저자에게 메일을 보냈었는데 응답이 없어서 학회장에서 직접 만나기를 벼루고 있었다. 정작 얼굴 맞대고 얘기하니, 다소 민감한 얘기라 안그래도 짧은 영어로 조심스럽게 물어보기가여간 어려운 일이 아니었다. AlphaRepair 대신 최신 모델을 쓰는게 우리 연구에 훨씬 도움이 될거라는 코멘트와 함께, 메일을 다시 보내주면 읽겠다는 아마도 지켜지지 않을 약속으로 만남은 끝났다.

멜버른 & 그레이트 오션 로드

다녀오기 전 호주는 별 생각이 없었다. 캥거루, 코알라 말고는 뭐가 유명한지도 잘 몰랐고, 인종차별도 심할 거라는 편견이 있었다. 내가 느낀 호주는 정반대로, 사람들은 정말 하나같이 다 친절했다. 멜버른은 중국인, 베트남인 등 여러 인종이 섞여살고 있었고, 도시는 세련되면서도 고풍스러운 양식을 갖춘 건물들의 조화가 참 멋졌다. 특히 학회 시작 전 하루동안은 그레이트 오션 로드 가이드 투어를 다녀왔는데, 호주에 간다면 꼭 시간을 내어서 다녀오라고 추천해주고 싶다.







마치며

사진 찍어준 한양대학교 모현민님 고맙습니다.

탑 컨퍼런스에서 활약하는 우리나라 연구자들을 보면서 자신감과 큰 동기부여를 얻을 수 있었다. 6년 전 처음 ICSE에 가서 느꼈던 분위기와는 정말 달라도 너무 달랐다. 다들 너무 잘해서 긴장감마저 느껴졌다. 여행도 즐거웠다. 같은 일을 하는 친한 친구 8명이서 해외에 나가는 경험은 대학원생이 아니면 인생에서 잘 겪기 힘든 경험일 것이다. 이것 저것 알아보느라 고생한 후배들, 현지에서 랩장 역할을 해준 명호형에게도 고맙다는 얘기를 전하고 싶다. 마지막으로 좋은 경험할 수 있도록 지원해주신 오학주 교수님께 감사드립니다.