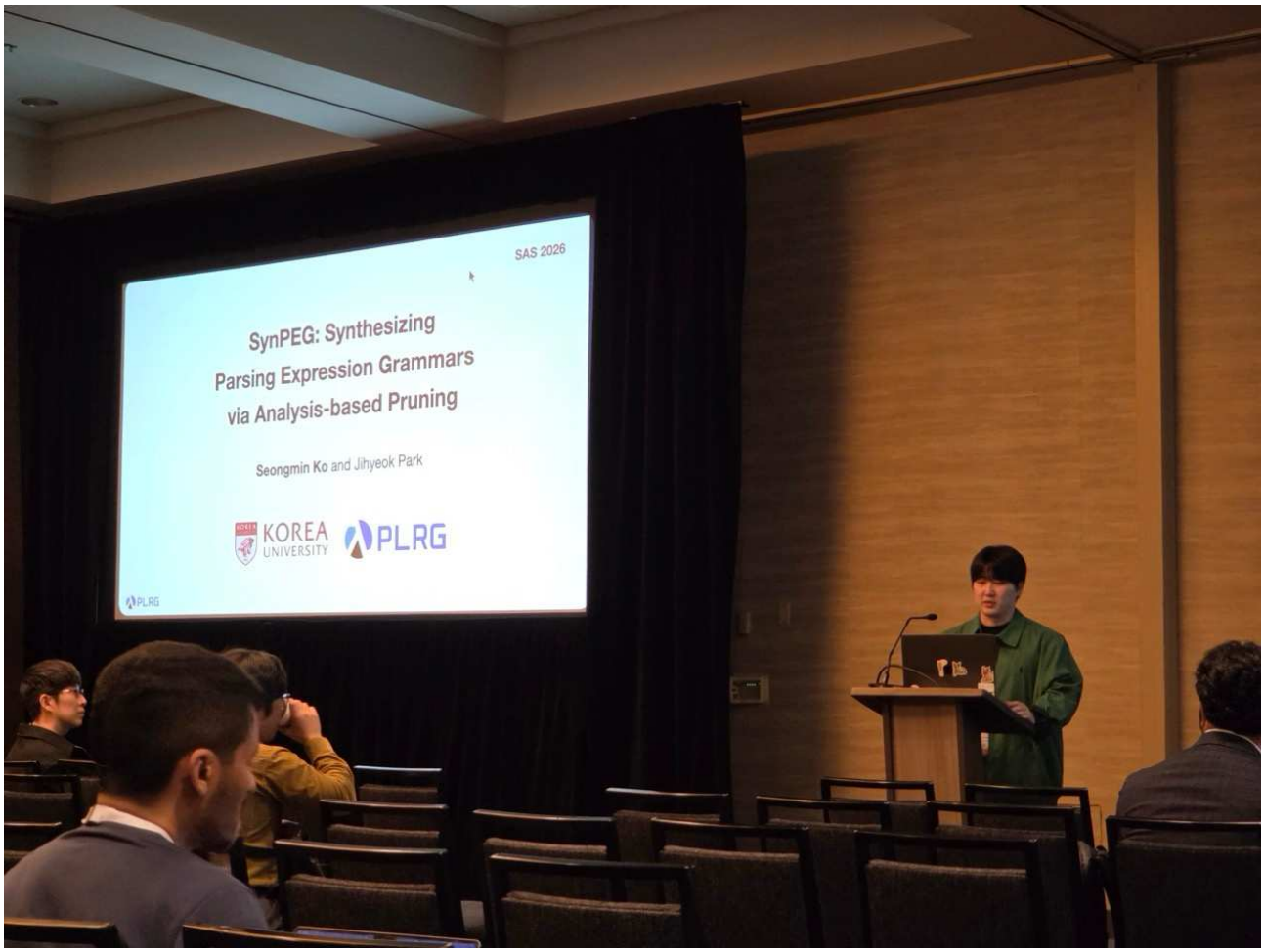
SPLASH 2026 Report

Seoul, Korea



Seongmin Ko, PLRG@Korea Univ.

5-8 October 2026

Introduction

👏SAS 2026 학회에 PEG Synthesis 논문이 accept 되었다.👏SAS 2026은 SPLASH/ISSTA 2026과 함께 열리기에 운 좋게 OOPSLA 학회도 참석할 수 있게 되었다. SPLASH/ISSTA with SAS 2026은 프로그래밍 언어 및 소프트웨어 공학 분야에 있어서 다양하고 획기적인 연구들이 많이 발표되는 최고 수준의 학회이다. 10월 4일부터 9일까지 미국 캘리포니아 오클랜드에서 열렸는데, 세 학회가 서로 겹쳐서 진행을 하였다. OOPSLA는 월-수, SAS는 화-목, ISSTA는 수-금 동안 열렸다.

해외 학회는 처음이었지만 작년 국내에서 개최되었던 PL분야 최고 학회인 PLDI 2025와 SE분야 최고 학회 중 하나인 ASE 2025를 들었던 경험으로 미루어 보았을 때 이번 학회는 재미있는 발표가 많은 학회였다. PLDI는 발표만으로 해당 연구를 이해하기가 어렵다는 점이 있었고, ASE의 경우 이해는 가지만 대부분 LLM을 방법론으로 사용한 논문이 대다수라 아이디어를 얻어가기가 어려웠다. 하지만 이번 학회는 발표를 통해 어느정도 연구의 방향과 의의를 이해가 가능하면서도 제시한 방법론들을 채택해볼 여지도 얻을 수 있었다. 학회를 다녀온 이후 연구실에서는 서로의 연구에 대해 여러 논의가 오가는 것을 보고 다녀오길 잘했다는 생각이 들었다. 또한 학회에 참석할 수 있도록 지원해주신 박지혁 교수님에게 감사의 말씀을 드리며, 이후의 이야기들이 학회 참석에 도움이 되길 바라며 글을 남긴다.

Presentation @ SAS

나는 수요일 15:30에 SAS에서 Synthesizing Parsing Expression Grammars via Analysis-based Pruning 논문을 발표하였다. 해당 연구를 석사 디펜스로 발표를 했기 때문에 발표 자료나 스피치가 문제가 없을 것이라 생각했지만.. 역시 해외 학회에서 발표를 한다는 건 긴장을 안할 수가 없었다. 그리고 SAS 발표 리허설을 학회 1주 전 진행하였는데, 디펜스 발표자료에서 벗어나 real-world 예제를 이용해서 새롭게 구성하여 발표하였다. 하지만 리허설 때 다시 롤백하자는 피드백을 받았다. 디펜스는 15분 발표였지만 SAS는 무려 30분(25분 발표, 5분 Q&A)이기 때문에 이것저것 내용을 더 넣다보니 난잡한 발표자료가 된 것이었다. 그래서 다시 흐름을 정리하고, 예제를 쉬운 예제로 챙겨가며 표현만 더 깔끔하게 다듬는 것으로 수정하였다. **무엇이든 본인 혼자만 고민하며 하다가는 local optimum에 빠지기 쉬우니 항상 남들에게 설명하고 이해시키며 이 연구를 모르는 사람은 어떤 것이 궁금하고 어떤 것을 이해시켜야 하는지를 파악하도록 하자.**

출국 1주전부터 학회날 저녁에는 계속해서 발표연습에 매진하였다. 그리고 영어로 25분 가량의 발표를 하는 것도 처음이었기 때문에 긴장하더라도 발표를 잘 할 수 있도록 연습하였다. 또한 수요일은 OOPSLA, ISSTA, SAS가 모두 같이 진행되는 날이기 때문에, 오후 유일한 synthesis 세션의 논문인 SynPEG 발표에 많은 인원이 참석할거라 생각하지 않았다. 하지만 직후 진행되는 business meeting에 참석하는 교수님들이 들으시게 되면서 꽤 많은 인원들 앞에서 발표하게 되었다. 너무 긴장한 탓에 28분 정도로 연습했던 발표도 너무 빨리 해서 20분만에 끝나버렸고, 질문에 대한 답변도 깔끔한 답변이 아니라 길어지는 답변을 해서 아쉽긴 했지만, 그래도 나름 성황리에 끝냈다고 생각한다. 질문 중 하나가 그래서 SynPEG를 어디에다 쓸 수 있냐는 실용성에 대한 질문이었는데, 이 부분은 학회에 붙었음에도 불구하고 계속해서 걱정하고 고민했던 부분이었다. 지금은 PEG를 주 연구로 하고있지는 않지만 필요한 부분이 생긴다면 언제든 다시 연구해보고 싶다.

Keynote

Mojo: Toward One Systems Language for All Hardware

Hardware를 위한 언어는 정말 어렵다. 특히 하드웨어마다 메모리, 연산량, 사용하는 instruction, 구조가 모두 다르기 때문에 사용자는 multi-layer로 코딩을 하게 되는데, hardware-aware하게 abstraction된 코딩을 진행하는 단계에서부터, 이후 본인의 하드웨어에 맞는 low-level 코딩을 하게 된다. 일반적인 유저라면 이미 많은 전문가들이 구현해놓은 library등을 사용하여 hardware-specific 코드를 작성할 필요는 없지만, 기업이나 더 뛰어난 성능을 원하는 개발자의 경우 어쩔 수 없이 해당 코드를 작성해야한다. 예를 들어 PyTorch로 ML 코드를 짠다면 PyTorch 개발자들이 만들어 놓은 optimizer로 CUDA 코드가 생성된다. 하지만 자신의 하드웨어 맞는 코드를 작성하고 싶으면 CUDA 코드를 직접 작성해야한다. 그래서 개발자들은 각 layer에 해당하는 언어를 공부해야할 뿐만아니라 layer간에 어떻게 연결되어있는지 파악도 해야한다.

이러한 불편을 해결하기 위해 하나의 programming 언어로 여러 하드웨어에 대응하는 코딩을 할 수 있도록 시작한 것이 Modular 사의 Mojo이다. 처음에는 단순히 Rust의 Ownership 개념을 CUDA에 접목한 정도라고 생각해서 흥미가 없었는데, Parametric Metaprogramming 부분이 아주 흥미로웠다. 사실 하드웨어 언어에 관심이 있었던 나는 예전에 각 하드웨어마다 최적화된 코드를 짜는 것이 아주 불편하니까 이걸 자동으로 합성하는 연구를 하려고 한 경험이 있다. 하지만 단순히 자신의 하드웨어 스펙을 입력하면 거기에 최적화된 CUDA 코드를 생성하는 AI 기업이 이미 너무 잘 하고 있었다. 그래서 CUDA 코드를 생성하는게 아니라 하드웨어 스펙 자체를 parametric하게 받아서 코드를 생성하는 방식으로 해서 CUDA layer를 아예 보지 않아도 되도록 만들어 보는게 어떨까 생각을 하기도 했으나 이쪽 도메인에 대한 무지로 인해 연구를 시작하지는 않았다.

하지만 Mojo가 그걸 하고 있었다! Mojo는 아주 강력한 metaprogramming을 제공한다. Parameter가 단순히 runtime에 받는 parameter가 아니라 compile-time에만 쓰이는 별도의 parameter도 지원한다. 예를 들면 matrix multiplication을 matmul[T, tile, layout, target](...) 처럼 구현하여 [...] 안의 파라미터를 이용하여 compile time에 specialization을 해 hardware specific한 코드가 실행되도록 하는 것이다. C++의 template과 다른 부분도 IR 단계에서 있다고 하는데, 이 부분에 대해서는 좀 더 알아봐야 할 듯하다. 한때 흥미를 가지고 보았지만 시도하지 못했던 연구주제가 진행되고 있는 것을 보니 AI가 발달한 지금은 나도 이해를 하고 기여를 해볼 수 있지 않을까 하는 마음이 드는 연구였다.

The Augmented Forest

Test-Driven Development의 저자 Kent Beck이 수요일 키노트로 augmented coding에 대한 소개를 하였다. AI가 아주 발달했고, 앞으로 더 발달한 현재 사회에서 우리는 AI를 어떻게 사용해야할까에 관한 발표였다. SNS를 보면 비전공자들도 자기만의 웹 페이지와 앱들을 만들면서 소개하고 자랑하는 글을 매우 많이 볼 수 있다. 이제는 code를 더 잘, 더 빠르게 만드는 것은 개발자의 역량이 아니다. 그것보다는 needs 파악, 확장성, 신뢰성 등을 고려하여 프로그래밍을 하는 것이 개발자가 눈여겨봐야 할 덕목이다. 많은 기능을 넣으려고 노력하지 말고, 많은 시도를 통해 필요한 것만 남기고 버리는 것을 중요히 여겨야 한다. 그리고 가장 크게 와닿은 부분은 AI를 혼자 학습하고 혼자 문제를 해결하고 하는데 쓰지 말라는 것이었다. AI를 반대로 누구와 소통할지, 어떻게 본인의 생각을 정리하여 남에게 소개할지에 사용하라고 이야기하였다. 예를 들어 버그를 발견하면 버그를 AI를 통해 이해하지 말고 해당 버그를 제일 잘 아는 사람을 AI에게 물어 사람간의 대화를 하라는 것이다! 앞으로 나도 혼자 연구를 AI와 진행하는게 아니라 연구실 멤버와 교수님과 나누면서 해야겠다는 생각이 들은 강연이었다.

Interesting Talks

Grammar Repair with Examples and Tree Automata



월요일 학회 시작 후 가장 처음으로 들은 연구 발표이다. 아무래도 PEG 연구를 했던 탓에 Formal Grammar에 대한 연구는 흥미가 생길 수 밖에 없는데, 이 연구는 ambiguous CFG를 unambiguous CFG로 repair하는 Greta에 대한 연구이다. CFG의 모호성에 관한 연구는 크게 1) 모호한지 판단하는 detection, 2) 어느 부분이 잘못됐고 반례는 무엇인지 찾는 localization & explanation, 3) 모호한 CFG를 수정하는 repair로 나뉜다. Greta는 모호한 CFG가 있을 때 OCaml의 Menhir parser generator가 생성하는 LR conflict 반례를 이용하여 CFG를 수정한다.

핵심적인 기여는 모호성으로 인해 발생할 수 있는 여러 parse tree를 유저에게 선택하도록 하고, 해당 선택한 예제를 기반으로 CFG를 수정하는데, 이해할 수 있을 정도로 수정한다는 점에 있다. 기존에도 tree automata를 이용하는 연구는 있었지만 그냥 repair를 하고 나면 사람이 이해할 수 없는 크기와 temporary한 nonterminal이 가득한 CFG가 결과로 나온다는 것이다.

이렇게 보면 Formal Grammar의 중요한 속성 중 하나가 '이해하기 쉬운가?'도 있다고 생각하는데, PEG가 그런 점에서 우수하다고 생각은 하지만 상용이 힘들다는 점이 아쉽기도 하다. 발표를 듣고나서 PEG를 확장하거나, 사용하기 더 쉽게 만드는 연구를 해볼까? 라는 생각이 들기도 했다. 하지만 그래서 PEG를 원하는 사람들이 누구인지, 어디에 적합하게 해당 도구를 쓸 수 있는지 동기가 부족한 상태에서 시작했다가는 SynPEG의 연구와 다를 바 없기 때문에 취미생활로 넘기기로 했다.

Reducing Hallucinations in LLM-Generated Code via Semantic Triangulation

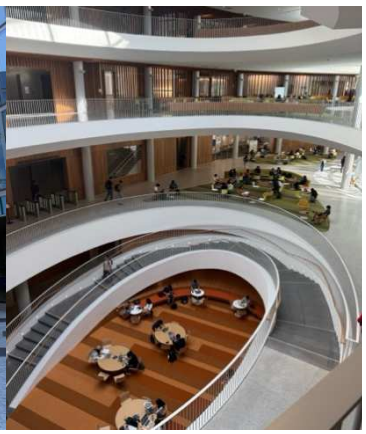


자연어 spec은 그 자체로는 모호성을 가지기 때문에 연구자들은 이 spec을 formal한 spec으로 변환하거나, 자연어 spec을 이용하여 여러가지 방식의 테스팅을 거쳐 모호성을 예방하는 연구를 하게 되었다. 이 연구는 후자에 해당하며, 단순히 여러번 생성하여 투표를 통해 가장 빈번하게 생성한 답변을 선택하는 것은 LLM의 데이터 의존성이나 hallucination의 영향을 받기 때문에 같은 문제를 풀게 하는 것이 아니라 논리적으로 반대되는 문제를 풀게 하여 두 문제의 답을 서로 비교하여 제대로 풀었는지 확인하는 방법이다.

예를 들어 encoding 코드를 만들고 싶다고 한다면 encoding하는 spec을 여러개 작성하여 여러개 생성해봤자 encoding이라는 단어에 편향되어 있는 LLM은 틀리지만 비슷한 답변을 계속 만들기 쉽다. 그래서 역문제인 decoding 코드를 만들도록 하여 encoding ↔ decoding 코드를 테스트를 통해 비교하여 코드가 제대로 작성된 것이 맞는지 확인하는 것이다. 두 코드는 역함수의 관계기 때문에 decode(encode(x))=x가 나와야 한다. 물론 역문제를 어떻게 생성할 것인지, 1 to n 함수의 경우 어떻게 할 것인지 등을 해결하기 위해 몇가지 방법들이 도입이 되었으며 발표에서는 자세히 다루지 않았지만 논문을 통해 확인할 수 있었다. 단순 LLM을 사용한 연구가 아니라 LLM의 문제를 해결하고자 하는 연구여서 재미있게 들었다.

Travel

@ UC Berkeley

목요일에는 Berkeley 대학에서 박사과정을 진행하고 계시는 이재호님의 연구실인 PLAITLAB의 학생들을 대상으로 교수님께서 강연이 있었다. 교수님께서는 ESMeta와 이번 ASE에 무려 distinguish paper로 붙은 Minipoly를 위주로 소개함과 동시에 개인미팅을 진행하셨는데, 그 동안 성민이와 나는 버클리 대학을 이리저리 구경하는 시간을 가질 수 있었다.

가장 충격적이었던 것은 CS 연구실들이 있는 건물인 Gateway 건물이었다. 재호님의 연구실도 이 건물 내에 있었는데, 외부에서 봤을 때 감탄이 나오는 것은 물론이고 내부구조가 정말 이공계 캠퍼스 희망편을 보는 듯 했다. 물론 우리 연구실이 있는 정운오 IT 교양관도 신축 건물이라 깔끔하고 좋은 건물인 것은 맞지만, 아무래도 서울의 땅값과 부족한 공간으로 인해 마치 빽빽한 회사 건물같은 느낌을 준다. 하지만 이 건물은 이공계 건물임에도 불구하고 라운지나, 연구실의 형태 등이 매우 마음에 들었다. 후에 알고 보니 준공이 2달도 채 안된 건물이라고 한다.

UC 버클리는 캠퍼스의 크기만 따지면 고려대학교와 크게 차이나지 않는 듯 했다. 하지만 조경이 매우 잘 되어있어서 캠퍼스 낭만을 즐기기에 아주 적합한 학교였다. 하지만 경사가 꽤나 높아서 실제로 재학중인 학생들은 걸어다니기가 매우 힘들어 보였다. 같은 코딩을 하더라도 사무실을 벗어나 나무아래 그늘 풀숲에 누워 작업하는 학생들을 보니 나도 학교 내에서 리프레시 할 수 있는 공간을 잘 찾아다녀야겠다는 생각이 들었다.




@ San Jose



학회가 시작되기 하루 전, 일요일에는 Oakland를 벗어나 San Jose를 탐방하였다. 낮에는 Levi's Stadium에 들러 San francisco 49ers와 Denver Broncos의 NFL 경기를 직관하였다. 마침 49ers가 현재까지 무패행진을 하고 있어서 사람들의 열기가 정말 대단했다. NFL 경기를 자주 봤던 것은 아니기에 전날 TV로 NFL 경기들을 보면서 어느정도 규칙을 이해했고, 그러고 보니 경기가 정말 재미있었다. 심지어 짜릿한 터치다운과 승리까지 가져가게 되어서 주변의 49ers 팬 아저씨들과 같이 응원도 하고 축하도 했다. 이후에는 산호세 지역에서 연구하고 있는 후배 친구와 함께 저녁을 먹으며 이곳저곳 둘러보게 되었는데, 정말 깔끔하고 안전한 동네라는 느낌을 받았다. 물론 모두가 차를 몰고 다니기 때문에 거리에 걸어다니는 사람은 많이 없었지만 동네의 분위기가 오클랜드와는 사뭇 다른 느낌이었다.