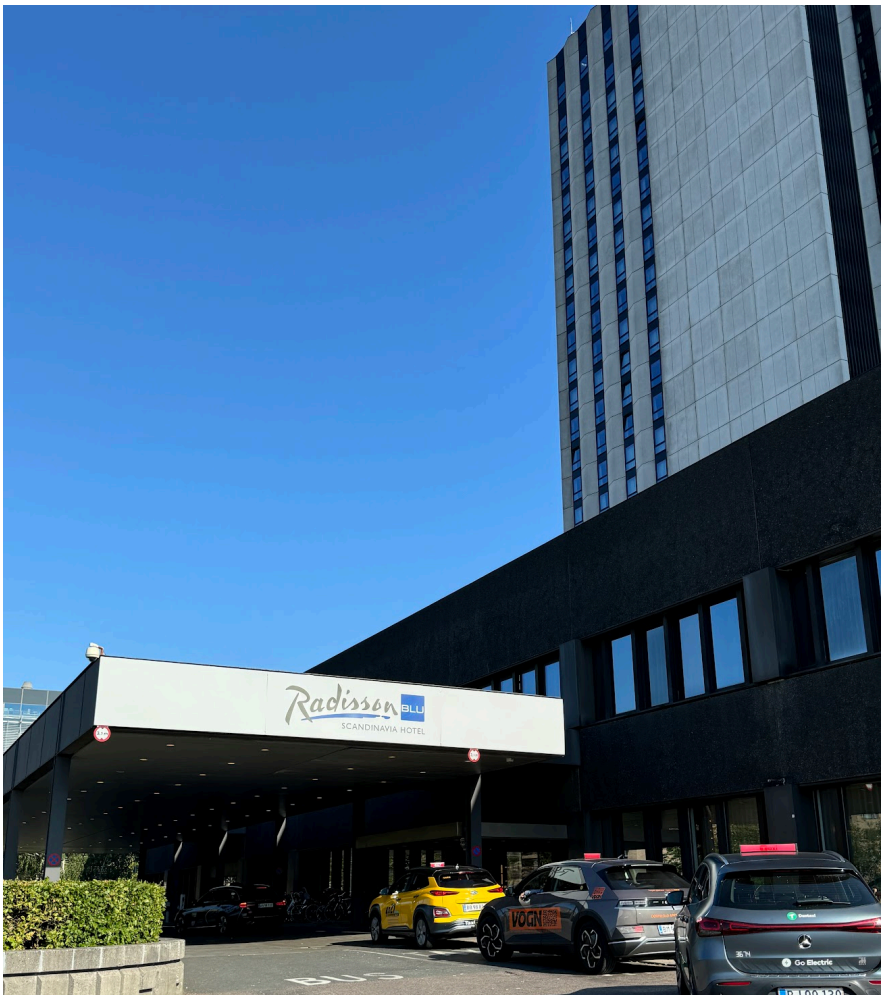
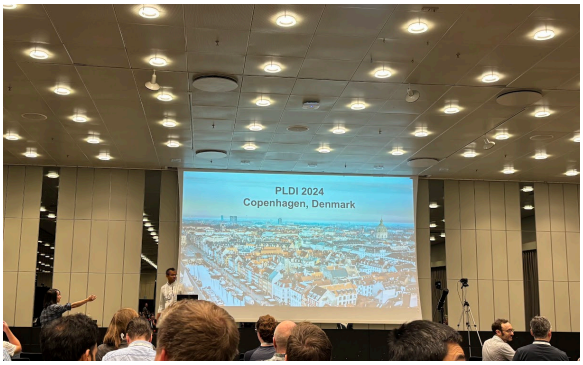
Trip Report for Attending PLDI24



2024. 6. 22. - 29.

최민석

PLDI (Programming Language Design and Implementation) 는 언어 설계 및 구현, 컴파일러, 프로그램 분석 및 최적화, 정형 검증 등을 다루는, 매우 권위 있는 연례 학술대회이다. ACM의 SIGPLAN 주관으로 이번에 덴마크 코펜하겐에서 45번째 학회가 열렸다. PL 분야에서 매우 중요한 학회이기 때문에 현장에서 참여하는 기회를 얻게 되어 매우 영광이었다. 직접 들으면서 인상 깊었던 몇가지 발표들을 소개한다.



PLDI24 Opening Keynote

Stream Types Stream Processing Program에 꼭 맞는 새로운 타입 시스템에 대한 발표였다. Stream Processing이란 입력을 한 번에 받을 수 없는, Incremental하게 입력되는 리스트를 다루는 컴퓨팅이다. 발표는 Stream Processing Programming을 위한 기존의 방법들이 프로그래머의 입장에서 얼마나 좋지 않은지를 설명하면서, List를 다루는 General Function Syntax로 Stream Processing Program을 작성하기 위한 방법으로 새로운 타입 시스템과, 이를 이용한 새 DSL, Delta를 제안한다. 발표에서 보여준 기존 Stream Processing Programming 방법의 단점 (프로그램이 One Big Combinator가 되어버리는 상황들)도 공감이 되었고, 이 문제를 깔끔하게 정의된 타입 시스템으로 풀 수 있다는 점과, 이로 인해 작성된 프로그램 코드의 형태가 매우 간결하다는 점이 우아해서 좋았다. "Rich types let us build stream programs compositionally" 라는 발표자의 강조가 매우 공감되었다. 마침 같은 날 키노트에서 DSL에 대한 내용이 나왔었는데, 'DSL에서 중요한건 domain보다 data structure'라는 키노트 발표와 겹쳐보이기도 했다.

[TOPLAS] Choral: Object-Oriented Choreographic Programming 이 발표는 TOPLAS (저널)에 게재된 내용을 PLDI에서 다시 소개하는 자리였다. 이 발표도 DSL에 대한 발표였는데, Distributed System을 위한 DSL을 소개하는 내용이었다. 발표에서 제시한 문제 상황은 Distributed System을 위한 프로그램을 작성하는 것이 어렵다는 점이었다. 이미 명세가 존재하더라도 이를 프로그램으로 옮길 때 component들의 다양한 State들을 각각 고려하면서 짜는 것이 어려울 수 있는데, 이를 해결하기 위한 방법으로 새 DSL을 제안했다. Java와 Interoperable하다고 해서 기존의 방법인 Akka보다 무엇이 더 좋은 건지 궁금해졌는데, Choral이 Akka 같은 도구보다 "훨씬" 더 boilderplate 코드가 적고 더 추상화된 프로그램을 작성할 수 있는 것으로 보였다. Choral에서는 정보의 위치를 타입으로 표시할 수 있는데 (e.g. int@A), 이를 이용해 정보의 전송같은 구현은 모두 숨겨지고 시스템의 choreography를 코드로 표현할 수 있다. 반면, Akka와 같은 도구에서는 각각의 Actor를 직접 작성해주어야 한다. 이 발표도 문제 상황에 맞는 DSL을 이용했을 때 얼마나 해답이 깔끔해지는지를 잘 보여준 것 같아서 좋았다.

Towards Trustworthy Automated Program Verifiers: Formally Validating Translations into an Intermediate Verification Language Automated Program Verifier의 Soundness를 보증하기 위한 연구였다. Translational Verifier는 프론트엔드에서 Source Language (C, Dafny, Viper 등)로 작성된 프로그램을 Boogie, Why3 등의 IVL (중간 검증 언어)로 번역한 뒤, 이를 백엔드에서 검증한다. 이 검증 결과를 믿을 수 있기 위해서는 프론트엔드와 백엔드가 모두 "번역 후의 프로그램이 correct하면 원래 프로그램도 correct"라는 soundness condition을 만족해야 한다. 이 연구는

현재 널리 사용 되는 translational verifier의 프론트엔드가 이론적으로는 sound함이 증명되었으나, 실제 '구현'도 sound한지 (즉, 구현 과정에서 이론상의 증명과 다른 구현 버그가 존재하는지) 검증된 적이 없는 것을 문제 상황으로 삼았다. 이를 해결하기 위해 Verifier를 수정하여 매 translation마다 Source Language와 IVL이 의미가 같은지 확인할 수 있는 "Certificate"을 독립적인 자동 증명기 Isabelle에서 돌릴 수 있는 형태로 출력한다. 흥미로웠던 점은 프로그램 번역 코드 전체를 직접 검증하지 않고 입력 프로그램마다 (동적으로) 동일성을 검증할 수 있는 증명을 자동으로 뽑아준다는 점이었다. verifier의 전체 코드가 너무 복잡하기 때문에 verifier를 직접 검증하는 대신 매 프로그램마다 동적으로 증명을 만들겠다는 접근인 것 같은데, 검증을 동적으로 생성한다는게 검증을 한다고 했을 때 떠오르는 이미지와는 좀 차이가 있는, 검증과 동적 테스팅을 잘 합친 아이디어 인것 같아서 재밌다는 생각이 들었다. 또한 번역 검증이라는 차원에서 소속 연구실에서 JS Minifier (Transpiler의 일종)를 연구하는 분도 있는데, minifier의 결과를 이런 형태로 검증하는 형태로 접목할 수 있지 않을까 하는 생각이 들었다.

Linear Matching of JavaScript Regular Expressions 이 연구는 오늘날 정규표현식 엔진들이 Backreference 같은 새 기능을 지원하기 위해 Exponential Backtracking을 사용하면서 ReDOS 취약점이 문제가 될 수 있다는 점을 문제 상황으로 삼고, JavaScript에서 사용하는 Regex 명세가 Linear 시간에 작동할 수 있는지 확인한다. 이를 위해 Regex match를 위한 널리 알려진 NFA JavaScript 명세에 맞게 확장하여 JS regex의 star 및 lookaround가 linear time에 mathcing 될 수 있음을 보이고 실제 V8 엔진에 기여했다. JavaScript는 원래 자신만의 이상한 semantic을 가진 것으로 유명했는데, 공교롭게도 lookaround rule도 다른 언어와 달리 특이하게 정의되어 있어 항상 선형 시간 안에 사용할 수 있도록 설계된 점이 신기했다. 또한 정규표현식이 ReDOS 문제를 일으킬 수 있다는 건 알고 있었고, 우리 연구실도 JS의 기계화 명세를 주요 기술로 가지고 있기 때문에 이런 연구를 어쩌면 직접 할 수도 있었겠다는 생각이 들어서, 왜 나는 이런 것을 떠올리지 못했을까, 하는 생각에 연구 주제를 찾는 법에 대해 고민해보는 계기가 되기도 했다.

Static Analysis for Checking the Disambiguation Robustness of Regular Expressions 마지막 날 가장 마지막 시간에 들은 발표이다. 정규표현식 엔진에서 많이 사용되는 두가지 disambiguation policy (POSIX & greedy) 가 있는데, 두 policy가 같은 정규표현식에 대해 서로 다른 match 결과를 낳을 수 있고, 이런 non-robust한 regex가 전체 데이터셋 중 4% 있음을 문제 삼았다. 이를 해결하기 위해 연구는 GNFA라는 NFA의 확장을 정의하고, 이를 탐색하면서 두 policy가 "new best match"를 찾는 선택에서 항상 일치하는지 확인한다. 탐색하는 robustness graph의 크기가 GNFA의 크기에 대해서 지수적으로 증가하긴 하지만 동일성을 유한한 시간 안에 확인할 수 있다는 점이 좋아보였다. 특히 연구진들이 몇가지 sufficient condition을 확인하도록 최적화하여 1시간 이상 걸리던 전체 데이터셋 확인을 3분 남짓한 시간에 완료할 수 있게 한 것이 인상적이었는데, 나도 개발을 할 때 정규표현식은 새로 짜지 않고 이미 존재하는 것을 재활용하는 경우가 많았는데, 문제 자체가 좋은 것 같아서 인상 깊었다.

소감 돌이켜 보니 PLDI에서 먼저 다른 연구자 분들에게 다가가지 못한 점이 아쉽다. 한국인의 밤이라는 자리가 있어 국내의 타 대학분들과 교류는 할 수 있었으나, 세계적인 학회인만큼 해외의 다른 연구자와 교류할 수 있는 자리인만큼 우리 연구실에서 앞으로 학회에 참석할 기회가 있는 분들께, 기회가 있을 때

마다 먼저 다가가 보라는 조언을 드리고 싶다. 이번에 운 좋게도 중간 쉬는 시간에 먼저 다가와주신 분들이 몇 있어 다행이라는 생각도 든다.
이번 PLDI 24에 참가하면서, 연구실에 들어왔던 동기를 다시 떠올리게 되었다. 연구실에 들어온 것 자체가 PL에 관심이 있어서였는데, 정작 들어오고 나서는 내가 잘 할 수 있을까 싶어서 SE 쪽으로 내심 방향을 틀었던 것 같다. PLDI에서 발표되는 재밌는 주제들을 보면서 다시 한 번 PL 쪽 연구에 길을 들이고 싶다는 생각이 들었다. 우선 이번에 참여하게 된 Partial Evaluation 연구 구현에 열심히 참여하면서 주제를 찾아보고 싶다.
이번 연구미팅 및 학회에 참석할 수 있게 해주신 박지혁 교수님과 여정에 필요한 여러가지 사무를 챙겨준 김준겸 선배에게 감사하다는 말을 남기면서 글을 마친다.