VMCAI2010 , POPL2010

Madrid, Spain, January 15~25, 2010

교수님, 영범이형, 순호형, 학주형과 Madrid에서 열린 VMCAI 2010, POPL 2010 학회에 참석하였다. 영범이형, 순호형이 대만의 왕 교수님과 쓴 논문이 VMCAI 에 붙어서 발표하러 가는 김에 견문을 넓히는 차원에서 나도 따라가게 되었다. 처음 가보는 POPL. 이 분야에서 날고긴다는 사람들의 발표와 대화를 들으며 마음을 새롭게 다잡을수 있었다.

VMCAI 2010

VMCAI논문의 후속논문 마감이 VMCAI발표와 겹쳤다. 순호형과 영범이형은 전쟁과 같은 3일을 보냈다. 학주형과 나는 상대적으로 여유가 있어 발표를 주의깊게 들을 수 있었다. VMCAI기간중에는 익숙한 주제를 다루는 논문들이 많아 발표를 따라가기가 수월했다. 익숙하지 않은 주제가 발표될 경우에는 아주 따라가기 어려웠다. 예전에는 기초지식이 부족해서 따라가지 못한다고 생각했는데 POPL에서는 모르는 주제가 나와도 대체로 따라갈만 했던것을 보면 꼭 듣는이만의 문제는 아닌듯 하다. 프로그램분석과 검증이라는 분야가 아직 덜 성숙하여 어휘가 제대로 통합되지 않은 탓일수도 있고 발표 방식의 문제일수도 있다. VMCAI의 경우는 두가지 다 해당되는듯 했다.

Matt Might,
Shape analysis of functinoal language

함수형 언어의 분석을 Shape Analysis의 어휘로 다시 설명했다. 개인적으로 함수형 언어의 분석에 관심이 있어서 특히 유심히 들었다. 환경은 본질적으로 heap에 할당된 그래프라서 어찌보면 당연한 내용이지만 발표자의 능력이 대단했다. 30분이란 제한된 시간 안에 어떻게 문제와 해결책을 직관적이면서 너무 추상적이진 않게 이야기하는지에 대한 한 모범이 될 만한 발표였다. 학회에서 돌아온 뒤 Matt Might의 논문들을 찾아 읽어봤는데 자신이 푸는 문제를 단순히 푸는 것이 아니라 보다 잘 표현하기 위해 끊임없이 노력하는 사람이라는 느낌을 받았다. 논문을 어렵게 쓰는 느낌이 들긴 하지만 아직 사골국을 끓이는 도중이라 그러려니. 국물이 완성되었을때 어떤 맛을 낼지 기대된다.

Roberto Giacobazzi,
Invited Tutorial : Abstract Interpretation-based Protection

Watermarking과 그에 대한 공격을 Abstract Interpretation관점에서 설명하였다. (AI를 이용한 Watermarking이 아니다) Watermarking을 파해치는 공격은 분석을 Complete하게 만드는 것과 동일하고 Watermarking은 분석이 바보같이 되도록 프로그램을 복잡하게 만드는 것 이라는 큰 그림 아래서, 어떻게 하면 Concrete Evaluation Coast를 작게 유지하면서 Analysis Coast를 비싸게 만드냐 하는 게임이 Watermaking에서 풀어야되는 문제라는 관점이 신선했다. 이런 통찰을 이용해서 구체적으로 어떻게 문제를 풀지 논의하지 않은 점이 아쉽기는 했지만, 기다리면 차근차근 나오겠지.

POPL 2010

POPL의 꽃은 타입인가보다. 타입에 대한 나의 공부는 Hindly-Milner 타입추론에서 멈춰있어서 그 이상을 다루고 있는 경우 디테일을 따라가는데 버거운 감이 있었다. 그러나 VMCAI때와는 다르게 무슨 문제를 어떤 방식으로 풀었는지 핵심은 잡아내기 어렵지 않았다. 발표자들이 그 부분을 전달하기 위해 혼신의 힘을 다하기 때문이었을까. 또 하나 재미있었던 것. 대가일수록 발표 내용이 간단명료하다. 예를 들자면 Andrew Pitts의 논문같은 경우 디테일을 발표하자면 장황하고 길어질수도 있는데, 발표에서는 문제상황과 그것을 위해 자신이 제시한 해결책에 대한 직관만 깔끔하게 전달하고 내려갔다. Swarat Chaudhuri같은 루키는 구현 디테일을 완전히 감추지는 못했지만 아이디어를 추상적으로 설명하기위해 발표를 갈고닦았다는 느낌을 강하게 받았다. 박사과정에 있는 학생들의 발표는 예제를 통해 하나하나의 아이디어를 이해시키기위해 많은 노력을 기울였음은 분명하지만, 모든 디테일을 설명하고자 하여 발표가 지루해지는 감이 있다. 일 하는동안 많은 시간을 쏟아 부었기에 하나하나의 디테일에 애정이 가는 것은 당연하다. 그러나 내가 애정을 가지고 있다고 해서 공중에게도 의미가 있다고 할 수 는 없다. 나의 관심이 아니라 듣는사람의 관심에 더 포커스를 두는것. 발표자로서 프로페셔널이 되려면 꼭 익혀야하는 스킬인듯 하다. 또 한가지, Dependent Type이 POPL의 뜨거운 감자인지 QA시간이 끓어올랐다. 동참하기 힘들어서 유감천만. 반성하고 공부에 전념할일이다. 내용을 따라갈 수 있던 발표중 기억나는 것들을 정리해보았다.

Program Synthesis

프로그램 합성을 주제로 세편의 논문이 발표되었다. 프로그램 합성에 대해 자연스럽게 두가지 질문이 떠오를 수 있다. 어떤 Constraint를 받으면 프로그램을 찾아낼 수 있는가? 어떻게 해야 프로그래머들이 Constraint를 자연스럽게 입력할 수 있을까?

Gulwani의 논문은 첫 번째 질문에 집중하였다. 프로그램이 실행중 가져야하는 제약사항을 Invariant로 제공하고 실행 후 예상 되는 결과는 Post-condition으로 제공하면 자동으로 프로그램을 합성해준다. "사용가능한 변수의 수" "재귀호출 가능여부" 등, Invariant를 자세하지만 높은 추상화 수준에서 주는것이 특징이다. 입력해야하는 Constraint의 양이 많아 실용성과는 거리가 있지만 앞으로의 발전방향을 주목해야될것 같다.

Bodik의 논문은 두 번째 질문에 집중하였다. 다년간 프로그램 합성 분야에서 일하면서 도달한 결론이 "Constraint입력의 편의성"인지, bodik은 논리식을 입력으로 받는 합성기가 사용자와 스무고개를 하면서 프로그램을 찾아가도록 하였다. "머리속으로 알고리즘의 큰그림만 잡으면 컴퓨터가 세부사항을 매꾸도록 하겠다"는 Sketch approac의 연장으로 보인다. 알고리즘 수업을 진행하고있는 교수들이 교육도구로서의 가능성을 높게 치는듯 했다.

마지막 Yorsh의 논문은 제한된 세팅에서 양쪽을 다 다루고 있다. 구체적으로는 CEAGAR 기반으로 Program을 분석하다가 진짜 에러를 만나면 그 에러가 발생하지 않도록 프로그램을 자동으로 수정해주는 기술을 다루고 있다. 논문에서는 Concurrent Program의 Race Condition을 예로 사용했는데, Path정보나 Context정보가 많이 필요하지 않은 버그를 잡을때 유용할것 같다. 세션 하나가 따로 할당되어 있는 것을 보면 프로그래머로부터 Constraint를 받아 그것을 만족하는 프로그램을 자동으로 생성해주는 Program Synthesis도 요즘 각광받는것 같다. 어떻게 보면 프로그램이 주어진 상태에서 프로그램이 실행중 가지는 성질을 추출해내는 프로그램 분석의 Dual이라 분석 기술이 발전하면 따라서 발전하는 것이 자연스러운것 같기도 하다.

Andrew Pitts,
Nominal System

lambda calculus에서 substitution이 lambda abstraction 안으로 파고들어갈 때에 binding variable을 새로운 이름으로 바꾼 뒤에 substitution이 파고들어가는 것을 알고있을 것이다. 이 논문은 "새로운 이름"을 쓰지 않는 의미구조를 제시한다. Multi-Staged language처럼 alpha-conversion을 사용할 수 없는 언어가 속속 등장하고 있는 터라 의미있는것 같다.

William R, Haris,
Program Analysis via Satisfiability Modulo Path Programs.

프로그램을 몇개의 Path program으로 나눠서 각각을 분석함으로서 정확도를 높이는 기법을 제시하였다. Slicing을 이용하는 분석은 새로운 아이디어가 아니지만 SAT solver를 이용해 분석해야할 최소한의 Path program을 자동으로 추출하는 부분에서 높은 점수를 받는듯했다. 파수에서 새로 만들고 있는 분석기와 많이 비슷해 보여서 깜짝 놀랐다. Trace Partitioning이 Trace를 BFS식으로 추적한다면 이 접근은 DFS이고, edge cutting을 잘 하는 경우라고 볼 수 있겠다. 문제공간 전체를 탐색해야하는 경우에는 DFS가 (메모리사용량에서) 효율적이듯, 분석도 더 효율적일수도 있을것같다

Swarat Chaudhuri,
Continuity analysis

주어진 프로그램의 동작은 입력이 아주 조금 변할때 어떻게 달라지는지 알아내는 분석을 소개하였다. 실수와 간단한 자료구조가 입력인 경우를 다루고 있다. 컴퓨터공학자들 보다는 제어, 시뮬레이션등응용을 개발하는 사람들에게 유용할 것 같다. 현장에 적용하려면 정수가 입력인 경우를 다룰 수 있어야 겠지만 , 값의 근사로 인해 발생하는 오동작을 예측할 수 있을테니. 문뜩 학계에서 프로그램 분석과 안전성 검증이라는 이름으로 다루는 내용이 작은 집단인 컴퓨터공학자만을 대상으로 하고 있는 것은 아닌가 하는 생각이 들었다. 컴퓨터를 사용하는 대다수의 사람들은 컴퓨터와 무관한 상위의 언어 (예를들어 전자기학이나 금융공학)로 시스템을 설계할것이다. 정말 필요한 것은 상위의 의미구조가 프로그램으로 번역되는 중간에 생기는 틈새를 찾아주는 것은 아닐지.

Joas Dias,
Automatically Generating Instruction Selector Using Declarative Machine Descriptions

Low level에서의 국소적인 최적화는 Binary 수준에서 바로 하는 것이 더 효율적이라는 전제하에 Machine Independent Optimization Compiler를 만들려면 어떻게 해야 하는지 다룬 논문이다. 구체적인 Instruction에 신경쓰지 않고 최적화된 3-Addresse Code로 변환하는데 집중해야 한다는 것이 골자이다. 각 추상화계층마다 최적화할 수 있는 성질이 다른것은 당연한일. 당연하고 누구나 알고있지만 구현하기는 어려운 것을 구현하는 것이 POPL에 올만큼 커뮤니티에 공헌이 된다는 사실이 놀랍다. 또 한가지 생각해 본 것은, 프로그램 분석의 계산을 최적화 하는데도 같은 통찰을 적용할 수 있지 않을지. 지금은 Abstract Semantics를 실행시켜주는 Interpreter를 바로 만들지만, 그보다 낮은 수준의 Abstact Machine으로 변환한다던지하는 식으로. 분명 Abstract Semantics만 가지고 계산을 최적화하는것과는 또 다른 이익이 있을 것이다.

톨레도

POPL이 끝난 다음날 하루 시간이 비어 영범이형, 학주형과 같이 톨레도를 구경하였다. 톨레도는 산위에 있는 도시로 과거에는 철벽요새로 이름을 날렸다고 한다. 도시의 남쪽으로는 강이 흐르고 북쪽은 급격한 경사면이 있어 한눈에도 천해의 요새로 보인다. 반면에 발전하기에는 어렵다. 아이러닉 하게도 도시의 발전을 가로막는 지형 덕에 고건물들이 손상되지 않고 남아있어 지금은 관광산업으로 먹고살고있다. 스페인 건축의 변화를 보기에 이보다 좋은 곳이 있을까 싶다. 톨레도를 둘러싼 성벽과 중앙의 대성당이 대표적이다. 유럽 배낭여행을 다녀왔다면 대성당에 첫 발을 디뎠을 때의 충격을 머리속에 간직하고 있을것이다. 톨레도의 대성당은 3세기동안 지어졌다고 하는데 마무리지어진 시기에 따라 구역마다 장식양식이 다르고, 특히 조각과 그림속에 사람을 묘사한 방식이 다양하여 보는 즐거움이 있었다. 다만, 성당 내부 공기가 쌀쌀하여 옷을 얇게 입고간 우리는 오래 머물지 못하고 나와야했다. 성벽은 12세기쯤 지어지고 무수히 개축되었는데, 개축된 시기에 따라 사용된 재료와 공법이 달라 문자 그대로 도시 역사의 퇴적층이다. 이 오래된 성벽 바로 뒤로는 주민의 편의를 위해 설치한 산 아래로 통하는 에스컬레이터가 다닌다. 약 9세기에 걸쳐 쌓이고 섞인 사람들의 흔적이 아주 인상적이다.

Wontae Choi