Trip Report

OPLSS 2010

June 14 - June 28, Eugene, Oregon, US

들어가기에 앞서 이 이야기는 포항공대 통합과정 박종현군의 경험을 토대로 작성하였음을 알려드립니다.

Oregon Programming Language Summer School (OPLSS)는 2002년부터 시작해서 올해로 9회를 맞이하는 깊이 있고 체계화된 교육 프로그램입니다. 올해는 Logic, Languages, Compilation, and Verification이라는 주제로 흥미진진한 강좌들이 많이 개설되었습니다. 본 글에서는 각 강좌들을 간단히 소개하고자 합니다. OPLSS 2010 홈페이지에 각 강좌에 대한 자세한 소개와 관련 자료들을 제공하고 있으니 참고하시길 바랍니다.

사진 1 강의가 진행된 Knight Law Bldg. 앞에서

이번 OPLSS에서는 총 40개의 80분 강의가 진행되었으며, 총 9분의 교수님들이 각기 다른 주제로 4~5회의 강의를 해주셨습니다. 아침 9시부터 오후 5시까지 매일 4개의 강좌가 알차게 진행되었습니다. 각 교수님들이 진행하신 강좌는 다음과 같습니다.

A. Type Theory Foundations - Robert Harper

Harper 교수님은 타입 이론(type theory) 연구의 기초적 도구 중 하나인 "proof by logical relation" 기법을 자세하게 소개해 주셨습니다. 타입 이론에 기초한 여러 함수형 언어에서 "well-typed programs always terminate!"이라는 성질이 성립한다는 것을 "proof by logical relation" 기법을 이용해서 직접 증명하시면서 설명해주셨습니다. 처음에는 잘 이해가 안되던 부분도 실제 증명을 보면서 설명을 들으니 나중에는 쉽게 이해할 수 있었습니다. 물론 뒷부분의 System F에 대한 termination 증명에서는 안드로메다로 여행을 떠났다가 지구로 간신히 귀환하는 쾌거(!!)를 이룩하여 개인적으로는 매우 뿌듯한 강의였습니다. 보다 자세한 내용이 궁금하신 분은 Advanced Topics in Types and Programming Language (Benjamin C. Pierce) Chapter 6을 참고하시면 될 것 같습니다.

사진 2 필자가 수업 중간 방문한 안드로메다 은하

B. Proof Theory Foundations - Frank Pfenning

Pfenning 교수님은 증명 이론(proof theory)의 기초에서부터 응용까지 방대한 내용을 깔끔하게 정리하여 소개해 주셨습니다. 이 수업을 통하여 다음과 같은 질문에 대한 답을 구할 수 있었습니다.

- Natural deduction은 무엇이고 이를 통해 논리를 어떻게 정의하는가?
- Curry-Howard isomorphism은 무엇이고 무슨 의미를 가지는가?
- Sequent calculus는 무엇이고 그것이 proof search에 있어서 어떤 의미를 가지는가?
- 보다 효율적으로 proof search를 수행할 수 있는 방법은 없는가?

특히, 최근에 자동 정리 증명기(automated theorem prover)와 관련된 연구를 수행하고 있었는데, 이와 연관된 내용이라 많은 도움이 되었습니다.

C. Dependently Typed Programming - Conor McBride

McBride 교수님은 dependent type을 지원하는 programming language에서의 programming에 대해서 소개해 주셨습니다. Type에 길이가 표현되는 Vector를 주 예제로 설명을 하셨는데요, dependent type을 이용할 경우 type을 program의 명세(specification)로 사용할 경우 보다 정확한 명세를 작성할 수 있다는 사실을 보여주셨습니다. 예를 들어, 두 개의 Vector에서 element 단위로 연산을 수행하는 함수의 경우 입력되는 Vector의 길이가 동일하다는 조건이 필요한데, dependent type을 지원하는 경우 이러한 요구 조건을 type에 명시할 수 있다는 점이 흥미로운 부분이었습니다. 특히 Haskell의 경우 명시적으로 dependent type을 지원하는 건 아니지만 type class를 이용해서 이를 시뮬레이션 할 수 있다는 점도 흥미롭더군요. 이외에도 Coq과 같은 Proof assistant인 Agda의 사용법을 소개해 주셨는데, Agda에서 지원하는 refine을 통한 programming 지원 기능이 유용해 보였습니다.. 수업 후반부에는 universe polymorphism에 관한 내용도 열심히 설명해주셨는데 너무 어려워서 잘 이해가 되지는 않았습니다. ㅠㅠ

D. Computational Type Theory - Robert Constable

Constable 교수님은 Coq 등에서 구현되어 있는 Calculus of Inductive Constructions (CIC)와 유사한 성격을 지니는 또 다른 이론인 Computational Type Theory (CTT)를 소개해 주셨습니다. 제가 이해한 바로는 컴퓨터 분야에서 사용되고 있는 type theory와 수학 분야에서 사용되고 있는 set theory를 연결시켜 consist한 이론을 만드는 것이 목표로 보였습니다. 내용은 흥미롭기는 했지만, 수업에서 다루는 주제가 너무 방대해서 이해하는데 어려움을 겪었습니다.

E. Programming Language for Compositional Security - Anupam Datta

Datta 교수님은 programming language를 이용해서 네트워크 프로토콜이 안전(secure)하다는 것을 증명하는 기법에 대해서 소개해 주셨습니다. 네트워크 프로토콜의 동작을 program으로 작성하고, 일반 사용자는 해당 program만 수행한다고 가정한 후, 악의적인 공격자는 임의의 프로그램을 작성해서 실행할 수 있다고 할 때 이러한 program들의 수행 과정에서 특정 성질이 만족된다는 것을 증명하는 기법을 설명해 주셨습니다. 이 기법의 장점으로는 기존 프로토콜들이 결합되어 새로운 프로토콜이 만들어지거나, 아니면 여러 프로토콜들이 동시에 네트워크 상에서 동작하고 있을 경우에도 기존 증명을 다시 사용할 수 있다는 점이었는데요, 이런 노력을 통해서인지 최근에는 네트워크 프로토콜 자체의 문제로 인해 보안 문제가 발생하는 건 본적이 없는 것 같습니다. 하지만 최근 문제가 되는 부분은 네트워크 프로토콜의 안정성을 증명할 때 가정하는 부분("비밀키는 특정 사람만이 알고 있다."와 같은 가정)을 공격하는 경우가 많아서 이 부분에 대한 대응이 중요할 것 같다는 생각이 들었습니다.

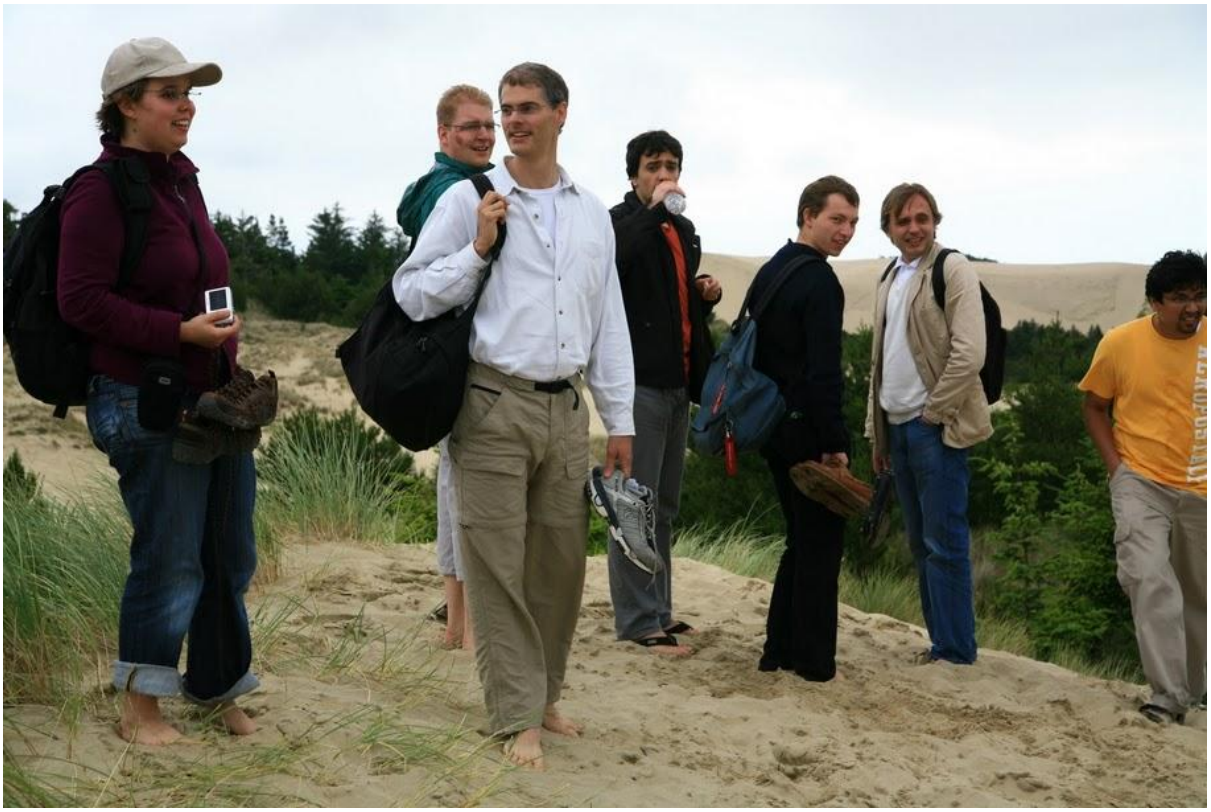

사진 3 주말에는 주변 해안을 방문하였습니다.

F. Proving a Compiler - Xavier Leroy

Leroy 교수님은 컴파일러가 의미를 보존하면서 프로그램을 컴파일한다는 사실을 어떻게 검증하는 방법을 주제로 강의를 해주셨습니다. 프로그램의 operational semantics를 정의하고 컴파일하는 과정에서 컴파일 전후에 operational semantics 상의 연관 관계가 있음을 증명하는 방식으로 증명을 수행하였는데요, 이 과정에서 프로그램의 operational semantics를 어떻게 정의해야 위와 같은 증명이 가능한지에 대해서 다양한 예제를 통해서 설명해 주셨습니다. 이후에는 컴파일 과정에서 사용하는 여러 최적화 기법들도 강의해 주셨는데요, 이러한 최적화에도 불구하고 컴파일 전후에 operational semantics의 연관 관계를 변하지 않는다는 것을 증명하는 법을 보여주셨습니다. 마지막에는 이러한 아이디어를 실제 대규모 컴파일러 작성에 도입한 CompCert 프로젝트의 내용에 대해서도 설명해주셨는데요, 아직 일부분에서 부족한 부분(예를 들어 parsing이 correct한가? Coq으로 모델링한 어셈블리의 모델이 correct한가? 등등 추가로 검증이 필요한 부분이 있는 것으로 보였습니다.)이 있기는 하지만 이러한 대규모 프로그램에 대해서도 검증이 가능할 정도로 Coq이 발전했다는 사실이 놀라웠습니다.

G. Ynot Programming - Greg Morrisett

Morrisett 교수님은 Coq을 이용해서 C와 유사한 imperative programming language에 대한 검증을 수행할 수 있는 도구인 Ynot을 소개해주셨습니다. 기본적인 아이디어는 imperative programming language에 대한 검증에 사용하는 Hoare Type Theory (HTT)를 Coq에 적합한 형태로 변형하는 것이라 생각됩니다. Ynot 자체는 Hoare Type Theory과 관련된 공부를 한 적이 있어

서 이해하기 어렵지 않았습니다. Ynot에 대한 내용 외에도 definition에 변화에 대해서 robust한 proof를 작성하는 방법에 대한 부분도 소개해 주셨는데요, 이에 대해서 어떤 분이 그렇게 작성된 proof는 robust하긴 하지만 너무 읽기 어려운 것 아니냐고 문제를 제기하셨는데요, 답변은 어차피 proof 잘 써놔도 읽는 건 어렵다 또한 Coq을 이용한 proof가 주어질 때 사람들의 주요 관심사는 증명한 theorem과 lemma지 구체적인 proof가 아니다 그러므로 읽기 어려워도 robust한 proof가 좋다는 식으로 말씀하시더군요. 이 부분에 대해서는 사람들의 의견이 분분했습니다. 그리고 수업 중간 중간에 Coq을 잘하려면 프랑스 Post-doc을 채용하면 된다는 썰렁한(?) 농담도 하셨는데요 왠지 프랑스 사람들이 좀 싫어하는 분위기더군요 ㅎㅎ

H. Software Foundations in Coq - Benjamin Pierce

Pierce 교수님은 Coq을 이용해서 다양한 programming language들을 formalize하는 방법에 초점을 맞추어 강의를 진행하셨습니다. 단순한 정수에 대한 사칙 연산만 가능한 언어, 간단한 imperative programming을 지원하는 언어, Simply-typed Lambda Calculus에 이르기까지 다양한 언어들에 대해서 Coq으로 이런 언어들을 모델링하고, 언어의 semantics를 정의한 후, 언어의 다양한 성질을 Coq으로 증명하였습니다. 이를 통해서 언어에 대한 meta-theory의 개념과 이를 Coq을 사용해서 formalize하는 방법에 익숙해 질 수 있는 좋은 기회였습니다.

사진 4 강의가 진행 중인 강의실 풍경!

I. Essential Coq from Scratch - Andrew Tolmach

Tolmach 교수님은 Coq의 기초에 대해서 강의를 해주셨습니다. Coq의 기능을 가장 기초적인 수준

에서 어떻게 사용하는지를 강의해주셨습니다. 흥미로운 점은 Coq의 core calculus는 생각보다 매우 작다는 사실이었습니다. 제가 이해하기로는 inductive definition과 implication 정도만 core calculus에서 지원하고 conjunction과 같은 부분은 이를 이용해도 유도하는 것으로 보였습니다. 내용 자체는 Coq의 매우 기초적인 내용이어서 평이했습니다. 수업에서 기억에 남는 부분은 Coq을 사용하는 이유가 무엇이나? 에 대한 질문에 대한 답하는 부분이었는데요 Coq을 trust base로 삼아서 proof를 개발해 나갈 수 있다는 것이 장점이라고 합니다. 즉, 전체 proof가 신뢰성 있는지 여부는 Coq이 신뢰성 있는지 여부만 점검하면 되는데, 이때도 Coq의 전체 시스템이 아니라 매우 작은 부분인 Coq의 type checker만 신뢰할 수 있으면 된다고 합니다. Coq의 tactic은 내부적으로 proof term을 만들어가는 방법을 제시해 줄 뿐이고 최종적으로 Qed가 입력되면 type checker가 proof term을 체크하게 되는데 tactic system은 방대해서 검증하기 어렵지만 type checker는 비교적 그 규모가 작아서 검증이 용이하다고 합니다.

총평 :

관심 있었던 분야들에 대해서 매우 유익하고 질 높은 강의들을 들을 수 있었던 정말 잊지 못할 행복한 기회였습니다.