Trip Report on Oregon Summer School

June 15-25, Eugene, Oregon, US

오학주 (pronto@ropas.snu.ac.kr)

1. 소개

OPLSS (Oregon Programming Language Summer School)은 미국 오레곤대학에서 매년 열리는 프로그래밍 언어 관련 여름학교이다. 나는 작년에 포항공대에서 참석한 것을 보고 처음 접했고, 또 올해 처음 참가하게 되었다. 유명한만큼 강사진이 화려하다. 올해에는 Robert Harper, Benjamin Pierce, Xavier Leroy, Frank Pfennig, Robert Constable 등 PL분야에 있으면 이름을 익히 들어볼수밖에 없는 사람들이 대거 참여 하였다. 많은 기대를 품고 오레곤으로 출발했다.

여름학교 장소는 유진에 위치한 오레곤대학이었다. 매년 이곳에서 열린다고 한다. 이번에 유진이란 곳을 처음 알게되었는데 포틀랜드에 이어 오레곤주의 두번째로 큰 도시였다. 하지만 포틀랜드와는 달리 매우 평화롭고 조용한 곳이다. 높은 빌딩은 거의 없고 시내 중심지와 그밖의 구분이 잘 가지않는 도시이다. 목재자원이 풍부한 오레곤주의 도시답게 큰 나무들이 도시 전체에 즐비하다. 오레곤대학도 매우 조용하고 여유롭다. 학교 곳곳에서 으리으리한 나무들과 그 위를 돌아다니는 다람쥐를 쉽게 볼 수 있다.

여름학교 프로그램은 꽤 빡빡했다. 80분길이의 40개 강의가 10일동안 오전 9시부터 오후 5시에 걸쳐서 진행되었다. 모두 9명의 교수님들이 강사로 참여했고 한 분당 4번 이상의 강의를 하셨

다. 내용도 호락호락하지 않았다. 보통 한 과목(4회분)당 반학기에서 한학기 분량의 내용이 압축적으로 진행되었다.

주 내용은, 부제(Logic, Languages, Compilation, and Verification)에서도 알 수 있듯이, 로직과 검증이었다. 더 구체적으로는 증명보조기(proof assistant)를 이용한 기계적인 증명과 관련한 내용이 많은 비중을 차지했다. 9개의 과목 중 4개가 Coq을 직접사용하는 과목이었다. 나의 경우에는 평소에 별로 접해보지 못했지만 중요하다고 생각해왔던 것들이어서 특히 관심을 가지고 들었다.

강사진은
http://www.cs.uoregon.edu/Activities/summerschool/summer10/speakers.html
강의스케줄은
http://www.cs.uoregon.edu/Activities/summerschool/summer10/schedule.html
에서 확인할 수 있다.
강의자료들은 http://sites.google.com/site/oplss10/ 에 정리되어 있고 여름학교 홈페이지 강의스케줄 페이지(위의 링크)에 강의 동영상이 올라올 예정이다.

아래에서, 강의자료중 강의노트와 슬라이드는 본래의 링크가 깨질경우를 대비해서 별도의 링크를 추가하였다.

아래에는 각 강의별로 진행되었던 내용을 큰 그림 위주로 정리해보았다.

2. 강의

Proof Theoretic Foundation (by Frank Pfenning

Frank Pfenning이 네 번에 걸쳐서 증명이론(proof theory)에 대한 강의를 하였다. 강의의 전체적인 내용을 정리하면 다음과 같다. (url은 강의노트의 주소이다)

1. natural deduction
   http://www.cs.cmu.edu/%7Efp/courses/15816-s10/lectures/01-judgments.pdf
   (local link)
   먼저 판단(judgement)과 명제(proposition)를 구분해서 앞으로 사용할 판단의 형태를 정의했다. A가 명제일때, A is true 라는 판단을 사용한다. 판단의 의미는 구조적으로

(constructive) 정의한다: 즉, 그 판단을 내리는데 있어서 증명이 필요한데 그 증명을 어떻게 만들어 나가는가가 의미이다. 이런식으로 각 논리 연결자(logical connective)별로 도입규칙(introduction rule)을 정의했다. 이 때 도입규칙으로부터 유추될 수 있는 사실은 제거규칙(elimination rule)으로 표현한다. 도입과 제거(introduction/elimination) 사이의 올바른 관계는 지역적 안전성/완전성(local soundness/completeness)를 증명함으로써 보인다. 안전성은 제거규칙이 너무 강하지 않음 (not too strong)을 의미하고, 완전성은 그것이 너무 약하지 않음(not too weak)을 의미한다. 또 함의관계(implication)을 위해서 가설을 포함하는 판단(hypothetical judgement)을 정의했다.

2. verifications & sequent calculus
   http://www.cs.cmu.edu/%7Efp/courses/15816-s10/lectures/02-pap.pdf (local link)
   http://www.cs.cmu.edu/%7Efp/courses/15816-s10/lectures/08-seqcalc.pdf (local link)
   증명과정을 증명식(proof term)을 정의한 후 표현하여 조립해 나가면 명제 A의 증명과정은 프로그램이 되고 A는 그 프로그램의 타입이 된다 (curry-howard isomorphism). 각 명제들이 어떻게 '사용'(use)되고 어떤 증명들이 '생성'(verification)되는지를 직접 드러내도록 판단을 정의할 수 있다. A↑는 A has a verification, A↓는 A may be used을 의미한다. 자연적 연연법(natural deduction)으로 정의한 의미대로 각 연결자(connective)들을 A↑와 A↓를 이용해서 다시 정의했다. 함의(implication), 참(truth), 거짓(falsehood)의 경우에 주의가 필요하고 나머지는 쉽게 정의된다. 이렇게 정의된 로직은 시퀀트계산법(sequent calculus)을로 표현할 수 있고, 둘 사이의 관계에 대해서 증명했다. 또한 자연적 연역법(natural deduction)과 시퀀트계산법(sequent calculus)의 관계도 보였다.
3. focusing
   http://www.cs.cmu.edu/%7Efp/courses/oregon-m10/04-focusing.pdf (local link)
   포커싱(focusing)는 반복되는 증명(redundancy)을 줄이는 기법이다. 시퀀트계산법(sequent calculus)을 이용한 증명과정에는 비결정성(non-determinism)이 많이 생기는데 포커싱을 통해서 이를 제거한다. Pfenning 교수님은 먼저 어떤 명제의 증명과정을 예로 보이면서 포커싱을 통해서 오직 하나의 검증(verification)을 가지게 됨을 보였다. 여기까지는 제약이 있는 식에 대한 (negative fragment) 대한 초점맞추기였고, 이를 일반적인 경우로 확장했다. 증명이론에 대한 배경지식이 없어서 뒤로 갈수록 이해하기는 어려웠지만 흥미로운 강의였다.

강의 내용에 대한 레퍼런스들은 http://sites.google.com/site/oplss10/frank-pfenning 에 정리되어 있다.

Essential Coq From Scratch (by Andrew Tolmach)

Coq의 기본기에 대한 수업이다. B. Pierce의 온라인책 “software foundations in Coq” (http://www.cis.upenn.edu/~bcpierce/sf/)의 첫 5챕터가 4번의 강의에 걸쳐서 진행되었다 (책의 나머지 부분은 B. Pierce가 강의했다). 이 책의 특징은 Coq과 프로그래밍 언어 수업이 동시에 진행된다는 것이다. 사용자 정의 타입, 함수들을 정의하면서 Coq으로 그것들의 성질들을 증명하는 식이다. 예를 들어, 리스트 타입을 정의하고, 리스트의 길이를 구하는 함수, 리스트를 뒤집는 함수를 정의했다고 하자. 그 다음 Coq으로 원래 리스트의 길이와 뒤집은 리스트의 길이가 같음을 증명한다. 프로그래밍 언어의 개념들과 그것의 엄밀함을 Coq을 통해서 자연스럽게 경험한다. 더불어 Coq도 자연스레 익히게 된다. Tolmach의 강의에서는 주로 귀납법으로 함수형 프로그램의 올바름을 Coq으로 증명하는 방법을 다루었다. 리스트, 다형타입, 고차함수, 로직등이 주요 내용이었다.

http://sites.google.com/site/oplss10/andrew-tolmach 에 강의내용이 정리되어 있다.

Type Theory Foundations (by Robert Harper)

Robert Harper는 타입이론의 대가이다. 강의를 시작하기전에 이번 여름학교가 예년에 비해 더 이론적이라고 하더니 이 강의를 두고 한 말인듯 싶다. 강의의 핵심내용이 논리적 관계(logical relation)를 이해하는 것이었는데, 두번째 시간부터 급격하게 지쳐갔다. 첫 시간에는 언어의 정적/동적 의미구조(static/dynamic semantics)를 정의하고 진행/보존(progress/preservation) 증명 등 익숙한 내용이 진행되어서 여유가 있었지만 프로그램이 종료함(termination)을 증명하기 시작하면서 강의의 흐름을 놓쳤다. 종료정리(termination theorem)이 자유변수를 포함하지 않는 형태(closed form)에 대해서 말하고 있는데 증명해야 할 것은 텀이 자유변수를 포함하고 있는게 문제였다. 따라서 좀 더 강한 성질이 필요하고 이를 만들어 나가는 과정을 (지나치다 싶을 만큼) 매우 엄밀하게 보여주었다. 이 증명에만 거의 두 강의시간이 쓰였다. Goedel의 system T와 Girard의 System F의 예를 통해 강의가 진행되었다. Girard의 “Proofs and Types”를 읽어보라고 했다. 타입이론의 고전으로 불리우는 책이라고 한다. 이 책은 인터넷에서 전체를 다운로드 가능하다.

역시 강의내용과 자료를 http://sites.google.com/site/oplss10/robert-harper 에서 찾을 수 있다.

Software Foundations in Coq (by Benjamin Pierce)

Andrew Tolmach에 이어 Benjamin Pierce가 Software Foundations in Coq의 나머지 부분을 강의했다. 이번에는 Coq자체보다는 응용편으로써, Coq보다는 프로그래밍 언어 측면이 더 강조되었다. 먼저 IMP 언어의 문법/의미구조를 정의했다. 그 다음 프로그램 최적화 루틴을 하나 보여주고 그것의 올바름을 coq으로 증명했다.

다른 PL수업에서는 이러한 성질을 손으로 증명하는데 비해, 기계로 증명한다는 것이 이 강의의 가장 큰 차이이다. 기계를 이용한 증명(machine-checked proof)은 생각보다 더 매력적이었다. 아무래도 손으로 한 증명은 뭔가 미덥지 않기 마련인데, 기계로 증명하니 일단 그런 느낌없이 깔끔하게 증명된 느낌이 든다. 그리고 생각보다 손으로 증명하는 과정과 크게 다르지 않아서, 익숙해지면 정리를 이해하는데에도 도움이 될 것 같았다.

언어 정의후에는 호아논리(Hoare Logic)로 넘어갔다. Assertion, hoare triple 등을 coq으로 정의한 후 그것들의 기본 성질들을 증명했다. 그 후 추론규칙(proof rule)들을 정리로 표현했는데 이 과정에서도 각 룰들이 왜 성립하는지 기계적으로 증명이 되었다. 그 후에는 여러 예제 프로그램들을 호아논리(hoare logic)로 검증하는 예제들을 보여주었다. 이것들은 증명이 꽤 길어서 쉽

지는 않다. 하지만 손 증명도 비슷함을 감안하고 그 과정의 어려움을 생각할 때 나빠보이진 않았다.

이후에는 단순타입 람다계산법(simply typed lambda calculus)을 Coq으로 정의하고 안전성등을 증명했다. 여기에 레코드(record)등을 추가하여 언어를 확장하였다. Robert Harper도 이 강의를 들었는데 Benjamin이 타입이론 등에 대해서 한 이야기에 대해서 틈틈히 보충해 주었다. 알고보니 Benjamin Pierce의 지도교수가 Harper였다. Software Foundations in Coq 나머지 모두를 6번의 강의를 통해서 마쳤다.

Benjamin Pierce의 강의는 군더더기 없이 깔끔했다. 깨끗한 발음, 빠르지도 느리지도 않은 강의 진행, 강의 내용의 신선함 등등. 이번 강의를 들으면서 생각한 것은, 우리나라 PL수업에서도 Coq과 같은 증명도구를 이용한다면 더욱 흥미롭고 엄밀하게 언어를 디자인하는 과정을 더욱 생생하게 느낄수 있는 수업이 되지 않을까 생각하게 되었다.

강의소개는 http://sites.google.com/site/oplss10/benjamin-pierce 에 있다.

Ynot Programming (Greg Morrisett)

Greg Morrisett이 4번에 걸쳐서 Ynot Programming이란 주제로 강의했다. Ynot은 명령형 언어의 정의 및 증명에 최적화한 Coq 라이브러리이다. Coq이 메모리반응이 없는(pure) 언어이고 함수들이 모두 전체함수(total function)이어야 하다보니 명령형 자료구조라든지 I/O, 무한루프를 기술하기에는 불편하다는 동기에서 출발했다고 한다. Morrisett교수님은 모나드를 이용해서 이러한 문제를 어떻게 해결하는지 보여주셨다. 여러 택틱(tactic)을 조합해서 사용하는 Coq증명 스타일 (Chlipala-style)에 대한 내용도 있었다. 이 스타일로 여러 가지 증명을 보여주었는데 따라가기는 어려웠다. Chlipala-stype로 증명하면 Coq의 단점 중 하나인 증명유지보수(proof maintenance)에 도움이 된다고 한다. 호아의 타입이론(Hoare type theory)을 예로들어 초기 버전의 Coq증명이 앞서 언급한 기법들을 사용하여 얼마나 간단해지는지를 보여주었다.

강의내용요약과 자료들은 http://sites.google.com/site/oplss10/greg-morrisett 에서 볼 수 있다.

Programming Language Methods for Compositional Security (by Anupam Datta)

네트웍 보안 프로토콜의 안전성을 로직으로 증명하는 것에 관한 강의였다. 보안 프로토콜은 디자인하기도 어렵고 그것의 올바름을 확신하기도 어렵다고 한다. 그래서 실제로 표준으로 쓰이는 프로토콜 중에도 버그가 종종있고 공격의 대상이 되기도 한다.

먼저 타입시스템 등 PL과 직접관련이 있는 영역에서만 로직을 보아왔는데 보안분야에서 로직을 적용하여 문제를 푸는 것 자체가 흥미로웠다. Protocol Composition Logic (PCL)이 그러한 로직 가운데 하나이다: 안전한 프로토콜 액션들이 이 로직의 공리와 추론규칙으로 표현된다. 프로토콜은 프로그램으로 표현되고 로직 시스템을 통해서 프로그램이 추론될 수 있다면 안전한 것이다.

강의는 세 부분으로 이루어졌다: 모델링 언어의 정의, 로직 정의, 부분 증명으로부터 전체 증명을 만드는 방법(compositional reasoning). 언어는 프로토콜의 각 동작을 표현할 수 있도록 간단하게 정의되었다. 프로토콜 동작으로는 보내기(send), 받기(receive), 암호화(encrypt)가 있다. 로직은 별로 특이할바 없어보이는 일차(first-order) 로직이다. 이 로직으로 어떻게 보안성질을 표현하는지 예를 보였다. 부분을 조합해서 전체증명을 만드는 방법(compositional reasoning)이란 S1과 S2로 이루어진 프로토콜을 S1과 S2의 증명을 조합해서 증명하는 것을 말한다. 이를 위해서 로직시스템을 크게 두가지 (trusted program invariants, interface)로 나누어 각각의 룰을 정의했다.

강의자료와 레퍼런스는 http://sites.google.com/site/oplss10/anupam-datta 에서 찾을 수 있다.
(local link: slide1, slide2)

Computational Type Theory (Robert L. Contable)

전통적인 PL분야 역사를 설명하며 시작되었다. Church부터 시작해서 중간중간 무슨일이 있었는지... 나중에 알고보니--Benjamin Pierce가 마지막 강의 도중 알려줬다--이번 여름학교 강사들이 다음과 같은 관계에 있었다.

Church → Kleene → Constable → Harper → {Morrisett, Pierce}
(A →B: B는 A의 제자)

알고보니 Constable은 아주 유명한 사람이었다. Kleene의 제자로 수학을 전산학으로 연결하는데 중요한 역할을 했다고 한다. 박사학위를 받은 사람이 무려 40여명이 넘는다. 이중에는 전산학 초대 박사도 여럿이다. 제자중에는 Robert Harper, Steven S. Muchnick 등 유명인물들이 많다. 여름학교의 주 강사들 (Harper, Pierce, Morrisett)이 모두 직간접적인 Contable의 제자인 셈이다.

강의는 주로 CTT(Computational Type Theory)에 대한 설명이었다. CTT는 한가지 형태의 텀으로 구성된 문법구조와 보통의 과정을 드러내는 의미구조(operational semantics)를 가지는 계산 시스템이다. Coq등에서 쓰이는 CIC(Calculus of Inductive Constructions)와 비슷하다고 한다. 내용을 알아듣기는 어려웠지만, 대가의 풍부한 경험에서 우러나오는 직관과 유머가 돋보였다. 원론적인 내용을 다른 시각에서 바라볼수 있게 해 주었다. Curry-Howard isomorphism에 대한 내용이 기억에 남는다. Constable은 이를 설명하면서 유머러스한 표정을 지으며 타입과 명제는 본래 동일한 것이지 자신은 구조적동일관계(isomorphism)라고 생각하지 않는다고 했다. 원래 같은 것을 두가지로 나누어서 생각한 다음, 다시 구조적 동일 관계를 또 다시 만들 필요가 있느냐는 것이다.

Proving a Compiler: Mechanized Verification of Program Transformations and Static Analyses (Xavier Leroy)

요즘들어 검증된 시스템 소프트웨어에 대한 연구가 많이 보인다. Xavier Leroy도 프로그램의 기계를 이용한 검증(mechanized verification)에 몰두하는 듯하다. 이번 강의에서는 컴파일러 증명에 대해 다루었다. 올해 POPL에서도 검증된 컴파일러와 DBMS논문이 각각 한편씩 발표되었었는데, 이중, POPL 컴파일러 관련 논문은 Xavier Leroy가 쓴 것이었다.

컴파일러 증명이란 주로 컴파일러의 뒷부분 즉 정적분석을 이용한 최적화 루틴이 올바름을 보이는것을 뜻한다. 컴파일러의 올바름은 변환(컴파일)된 프로그램이 소스프로그램과 의미가 동일한지를 확인하는 것이다. 컴파일러 앞단(front-end)이 올바른가에 대한 이슈도 있지만, 그것보다 더 중요한 것은 컴파일러 최적화로 인한 것들이다. 최적화로 인해서 소스프로그램에 담겨있던 문법적인 구조가 많이 변형되기 때문이다. 문법구조가 변형되어도 프로그램의 의미가 그대로 보존되는가가 컴파일러 증명의 핵심이다.

지금까지는 컴파일러의 최적화 루틴이 엄밀히 또는 충분히 검증되지 않았었다. 컴파일러 최적화에는 전통적으로 데이터 흐름 분석(dataflow analysis)이 이용되는데 이 방식은 분석의 안전성을 엄밀하게 증명하는데 초점을 맞추기 보다는 효율적인 분석기법에 초점을 맞추어 발전하여 왔다. 간단한 분석인 경우에는 직관적으로 옳다고 넘어갈 수 있겠지만 언어와 분석이 직관적이지 않을때에는 그 올바름을 엄밀히 증명하기 어렵다. 반면, 요약해석(abstract interpretation)을 충실히 따라가면 안전성을 엄밀히 증명할 수 있다. 하지만 이 경우에도 종이에 증명한 것과 실제로 컴퓨터로 구현된 것 사이의 간격은 언제나 존재하기 마련이다.

이 강의에서의 컴파일러 증명은 증명할 성질을 엄밀히 증명하고 또한 그것과 구현의 간격이 없도록 하는 것을 뜻했다. Xavier Leroy는 간단한 컴파일러와 최적화 루틴에 대해서 기계적으로 올바름을 증명하는 과정을 보여주었다. 증명과 구현의 간격을 없애기 위해서 Coq으로 변환과 최적화 부분을 구현하고 그 루틴들의 올바름을 역시 Coq내에서 증명한다. 이 방식이 기존의 방법들과 달리 독특한 점이라 할 수 있다. 이를 “software-proof codesign”이라고 불렀다.

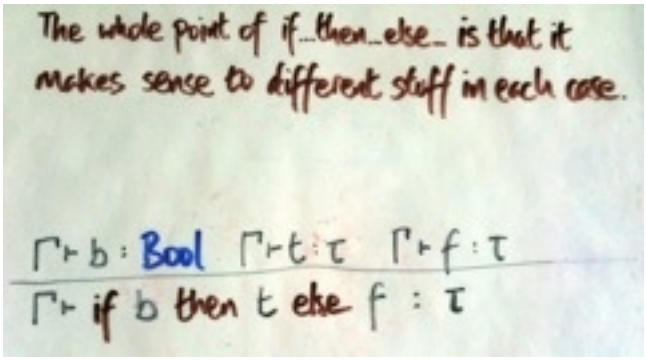
강의의 전체적인 흐름은 다음과 같았다. 먼저 소스언어(IMP)와 타겟언어(VM)의 정의와 변환 룰을 정의했다. 다음은 이 변환이 올바름을 보일 차례다. 관찰할 수 있는 행동들(Observable behavior)을 기준으로 두 언어로 짜여진 프로그램이 같음을 보이는데 보통 프로그램의 실행을 종료, 에러, 발산 으로 나누어 비교한다. 이 때 종료된 실행인 경우에는 마지막 결과값을 비교함으로써 같은지를 확인한다. 이를 관찰된 동일성(observational equivalence (bisimulation))이라고 한다. 이 때 한가지 제약을 두는데 비결정적인(non-deterministic) 계산순서는 특정 순서로 고정시킨다. 그 이유는 계산순서가 일정치 않은경우에는 증명하기가 매우 어렵거나 불가능하기 때문이다. 변환된 언어사이의 동일함을 바로 보이기가 어렵기 때문에 순방향/역방향 시뮬레이션(forward/backward simulation) 두 개를 보임으로써 같음을 보인다. 순방향 시뮬레이션(forward simulation)은 변환 프로그램의 성질은 소스 프로그램의 성질의 일부임을 보이는 것이다. 역방향 시뮬레이션(backward simulation)은 변환 프로그램은 소스프로그램의 성질을 포함함을 보인다. 이 때 중요한 것은 에러와 발산 상태를 구분하기 위해서 큰 보폭의 coinductive 의미구조(coinductive big-step semantics)를 사용하여 의미를 정의한다는 것이다. 보통의 의미구조(big-step semantics)로는 이 두 가지 상태를 구분하지 못하기 때문이다.

이 밖에도 최적화와 요약해석의 올바름을 보이는 과정도 있었다. 최적화의 예로는 liveness 분석을 이용한 dead code elimination을 예로 들었고, 요약해석으로는 구간도메인(intervals)을 이용한 간단한 요약해석기를 예로 들었다.

강의자료는 http://gallium.inria.fr/~xleroy/courses/Eugene-2010/ 에 정리되어 있다. (local link: slide, coq-files)

Dependently Typed Programming (Conor McBride)

독특한 강의였다. 첫 시간은 강의라기 보다는 OHP필름과 각종 소품을 이용한 쇼를 보는 것 같았다. OHP필름은 모두 손수 쓰고 그린 것들이었다. 앞으로 설명할 것들을 동기부여 시키기 위해서 if then else 구문의 타입룰을 보여주면서 then과 else는 서로 다른 일을 하기 위한 것인데 타입룰은 오히려 반대로 서로 같은 것(타입)을 가정하고 있지 않냐고 물어보셨다 (아래 그림). 맞는 말이었다. 그 다음에 하스켈을 이용하여 의존타입을 이용한 프로그래밍(dependently typed programming)에 대해서 벡터(vector)와 행렬(matrix)예제를 이용하여 설명하였다.



강의는 주로 Agda를 이용하여 진행되었다. Agda라는 언어를 처음 보았는데 값에 의존하는 타입(dependent type)을 지원하는 함수형 언어라고 한다. Agda를 이용하여 벡터(vector)등 기본적인 자료구조를 정의하는 방법부터 시작해서 단순 타입 람다계산법(simply typed lambda calculus)까지 예들을 보였다. 얼핏 이러한 의존타입시스템은 비용문제 등으로 실용적으로 쓰이기 어려울것이란 생각을 하고 있었는데, 쓸만한 수준의 언어가 있음을 알게 되었다. 하지만 Agda에 익숙하지 않은 상태였고 강의가 상당히 빨리 진행되어서 따라가기 어려운 강의중 하나였다.

강의 자료는 http://personal.cis.strath.ac.uk/%7Econor/Eugene/Conor/ 에 있다.
(local link: Coq files)

3. 맺음말

많은 것을 배울수 있었던 시간이었다. 모든 내용을 따라가긴 어려웠지만 평소에 잘 알지 못하던 분야에 좀 더 친숙해진 느낌이다. 다른 나라 학생들은 어떻게 공부하는지에 대해서도 체험할 수 있었다. 적극적으로 질문하고 토론하는 모습, 강의 후 강의실에 남아서 스터디 그룹을 통해 교류하는 모습이 인상적이었다. 아침식사시간에도 식당에서 노트북으로 Coq을 연습하는 사람들도 꽤 많이 보였다. 오레곤의 여유로운 모습도 기억에 오래 남을것 같다. 좋은 기회를 주신 이광근 교수님께 감사드린다.