Trip Report - PLDI 2014

2014년 6월 9일 - 13일
영국, 에딘버러

서울대학교 이우석

1. 개요

이번 PLDI는 여러 위성 워크샵들과 함께 6월 9일부터 6월 13일까지 영국 에든버러에서 열렸다. 나는 논문저자는 아니지만 감사하게도 센터 지원을 받아 학회에 참여하여 최근의 연구동향을 살펴볼 수 있는 기회를 얻었다.

에든버러는 바닷가에 위치한 도시로 유네스코에 도시 전체가 문화유산으로 등재된 곳인데, 몇백년된 구 시가지가 잘 보존되어있다. 옛날 양식의 건축물들이 흐린 안개속에 호젓하게 서있고, 갈매기가 날아다니는 하늘 아래로 백파이프 연주소리가 울려퍼지는 그런 곳이다. 호그와트 마법학교가 있어도 이상하지 않을 이런 곳에서 최신 컴퓨터공학 연구성과가 논의되는것이 묘했다. 도시가 예뻐서 자꾸 학회보다 다른곳에 관심이 가는것을 다잡느라 힘들었던 것이 이곳의 학회장소로서의 단점이라면 단점이다. 하지만 다행히 도시가 작아서 얼마 지나지 않아 금방 학회에 집중할 수 있었다.

이번 PLDI 참여는 공저자로서 참여한 2012년 PLDI에 이어 두번째인데, 그 때와 비슷하게 각자 주변의 문제를 아주 상식적인 방법으로 풀었다는 인상을 주는 연구결과가 많았다.

특별히 두드러지는 경향이라고 느꼈던 것이 있다. 결정론적인 알고리즘에 의해 정확하게 계산하는 것이 아닌 다른 철학에 기반한 방법들 (근사적 계산 (Approximating computation), 확률적 프로그래밍 등)에 대한 고찰이 많았고 이에 대한 사람들의 관심 또한 뜨거웠다. 근사적 계산은, 계산결과는 정확해야 한다는 믿음을 무너뜨림으로써 여러가지 이점들을 얻는 방법이다(계산 속도, 에너지 절약 등). 예를 들면, 그래픽, 오디오 관련 알고리즘 등은 인간의 감각이 차이점을 못 느낄 정도까지는 값을 부정확하게 계산해도 된다. 이에 대한 기조연설이 인상적이었다. 또한, 값이 확률적으로 결정되는 확률적 프로그래밍에 관한 논문들도 눈에 띄었다. 한 예로 확률적 프로그래밍에서 확률적 성질에 대한 assert 구문을 지원하는 방법에 대한 연구가 인상적이었다. 프로그램에서 오류 혹은 예외가 발생할 경우 이를 그냥 두루뭉술하게 넘어가는 방법만으로도 많은 프로그램들이 그럭저럭 기능을 할 수 있다는 조사를 한 연구도 있었다. 이런 연구결과들의 발표와 더불어, 근사계산에 대한 SIGPLAN 워크숍 APPROX가 첫번째로 이번 PLDI와 함께 열렸고, 사람들의 관심도 많았다. 거기서는 근사계산이 앞으로 지향해야 할 방향이 논의되었고, 그 중에는 프로그램 정적 분석에 관해서는 분석의 안전성에 대한 집착을 버리고 확률적으로 안전성을 보장하는 방향으로 가보자는 의견도 있었다.

그 외에 컴파일러 검산, 테스팅을 이용한 병렬 프로그래밍 지원, 정적분석의 요약 중 가장 비용효율적인 것을 찾는 방법 등 다양한 문제를 비교적 난해하지 않은 방법들로 푼 연구들이 많이 소개되었고 이들 중 인상깊었던 것들을 소개하고자 한다.

2. 발표된 논문들

What exactly is inexact computation good for?

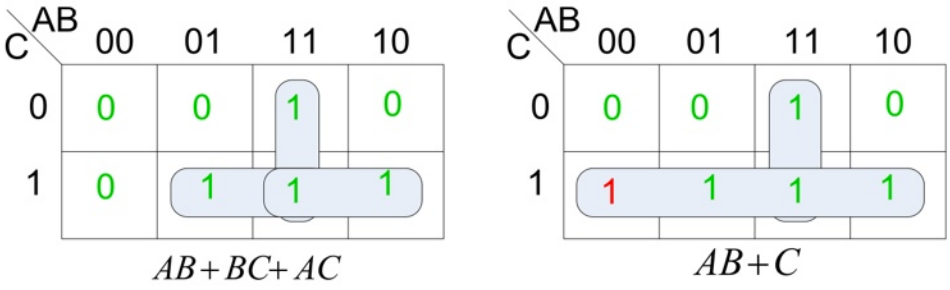
Krishna Palem (Rice University)

이 발표는 최근까지의 근사적 계산 관련 연구 성과를 총망라해서 보여주는 발표였다. 우선 이 발표의 제목에 대한 답은 눈치 채지 못할만큼의 에러율을 떠안는 대신 에너지 소비량을 줄일 수 있다는 것이다. 오디오/비디오 등 산출물이 사람의 감각기관에 의해서 인식되는 소프트웨어의 경우, 에러율이 심하지 않으면 사람이 인식하기에는 문제가 없는 경우가 많다.

이러한 근사적 계산은 프로그램의 명세를 완화시키거나, 하드웨어를 근사적 계산을 하도록 바꾸는 방식이 있다.

하드웨어를 바꾸는 한 예는 발표자가 속한 연구그룹에서 개발한 PCMOS (Probabilistic CMOS)라는 것이다. 이것은 기존의 CMOS와는 다르게 구현되어있는데, 특징은 값이 확률적으로 결정된다는 것이다. 그리고 값이 정확하게 결정될 확률은 에너지 소비량과 반비례한다. 이것을 이용하여 설계한 한 영상처리 회로가 8% 정도의 에러로 기존보다 4배이상의 에너지 절약을 이루어냈다고 발표자는 설명했다.

스펙을 바꾸는 한 예로는, 기존 회로보다 게이트를 덜 사용하지만 결과는 엇비슷한 회로를 사용하는 것이다. 예를 들어 다음과 같다.



왼쪽의 k-map을 오른쪽으로 바꾸면 8개의 입력 케이스 중 1케이스에 대해서만 결과가 다른대신에 사용되는 게이트 숫자는 기존의 5개에서 2개로 줄일 수 있다.

그렇다면 어느정도까지 정확도를 낮출 수 있을까? “충분히 정확하다”는 개념은 어떻게 정의될 수 있을까? 발표자는 사람의 뇌가 정보를 어떻게 인지하는지를 모델링하여 이로부터 충분히 정확하다는 개념을 정량화하여 시스템의 정확도를 결정짓는 방법을 고안했는데 이를 sensoptimized system이라고 부르며 소개하였다.

이렇게 구현된 시스템의 예들도 보여주었는데 주로 그래픽 쪽 응용들이다. 완화된 제약조건에서 구현된 ray tracer등의 시각화기는 겉보기에는 원래의 것과 별 차이가 없으면서 에너지 소비량을 줄일 수 있는 것으로 나타났다.

이번 PLDI 발표 들 중에서 가장 인상깊고 재미있게 들었던 발표이다. MIT Tech Review에서 세상을 바꿀 기술 10가지 중 한개로 뽑힌 바 있다는데 과연 그럴만 한 것 같다. 하지만 어플리케이션이 그래픽이나 오디오 관련 분야로 한정될 수 있다는 한계 또한 있으므로 응용분야를 잘 찾는것이 앞으로 중요하지 않을까 싶다.

Expressing and Verifying Probabilistic Assertions

Adrian Sampson, Pavel Panchekha (University of Washington), Todd Mytkowicz, Kathryn S. McKinley (Microsoft Research), Dan Grossman, Luis Ceze (University of Washington)

확률적으로 값이 정해지는 프로그램에서 assertion 구문을 넣는다면 어떠한 형태가 될까? 예를들어 다음과 같은 코드가 있다고 하자. 현재 위치에 랜덤 노이즈를 더해서 반환하는 코드이다.

```
def obfuscate_location(location):
noise = random.gauss(0,1)
d = distance(location, location + noise)  assert(d < 10) ??
return location + noise
```

이 때 noise의 절대값이 10보다 작다는 것을 확인하고 싶다. 노이즈는 랜덤하게 선택되는 값이므로 절대적으로 10보다 작다는 것은 보일 수가 없다. 그러나 확률적으로는 보일 수 있다.

이 코드를 작성한 사람이 위의 assert 부분을 다음과 같이 고칠 수 있다.

```
passert (d < 10, 0.9, 95%)
```

위의 것이 의미하는 바는 d 가 10보다 작을 확률이 90%라는 걸 95%의 신뢰수준으로 증명하라는 것이다.

위와같은 assert구문을 확인하는 방법은 예를 들면 다음과 같다. 예컨대 평균 1, 표준편차 3인 정규분포를 따르는 확률변수 x가 있고, x가 5 이상일 확률에 대한 성질을 확인하고 싶다고 하자. 다음 Chebyshev 부등식을 사용한다.

$$\Pr[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}$$

평균, 표준편차 값을 위 식에 대입하면 k가 2일때 x가 5이상일 확률을 구할 수 있다는 것을 알 수 있다. 위 식에 값을 적절히 대입해보면 결과적으로 x가 5 이상일 확률은 1/4 이하라는 것을 알 수 있다.

이러한 과정을 통해 정적으로 passert 구문을 바로 보일 수 있으면 성공, 아니면 샘플링을 한 후 hypothesis test 를 한다(passert의 인자로 들어오는 신뢰수준값이 이때 필요함).

이 연구는 프로그램 성질이 확률적으로 만족되는지 여부를 확인하는 새로운 방법을 제시했다는 점에서 의미가 크다고 생각했다. 또한 논문도 구조, 표현 측면에서 상당히 잘썼다는 생각이 드는 논문 중 하나이다.

Automatic Runtime Error Repair and Containment via Recovery Shepherding

Fan Long, Stelios Sidiroglou-Douskos, Martin Rinard (MIT EECS & CSAIL)

발표논문에 대해 1분동안 소개하는 자리에서 Martin Rinard 교수는 이 논문에 대해서, 절대로, 영원히 에러를 경험하지 않을 수 있는 방법을 알려주겠다고 했는데 문자 그대로 과연 그러했던 논문이다.

방법은 매우 간단한데, 에러가 발생하지 않게 예외처리 부분을 조작하는 것이다. 널 포인터 접근이나 0으로 나누기 에러가 발생하면 인위적으로 예외를 발생시키지 않고 0을 반환하게 한다. 그리고 그 상태로 프로그램 실행이 계속 진행되게 한다. 반환된 0 값에 의해서 연쇄적인 널 접근/0 나누기 오류가 발생할 수 있는데, 이를 방지하기 위해서 반환된 0 값에 대해 일어나는 모든 시스템 콜을 차단하고 아무일도 발생하지 않게 놔둔다. 그러다보면 프로그램이 자체 내장하고 있는 회복 루틴에 의해서 필요 부분이 다시 깨끗하게 되고, 다시 올바르게 진행된다. 이 논문이 가정하고 있는 핵심전제는 프로그램 자체가 가지고 있는 회복 루틴이 발동될 때 까지만 버티면 모든게 괜찮아진다는 것이다.

문자 그대로 에러를 나지 않게 하니까 에러를 경험하지 않을 수 있다. 18개의 CVE DB에서 추출한 에러경우에 대해서 이 방법을 적용했더니, 17개 경우에 대해서 개발자들이 오류 수정을 한 결과와 겉으로 보이는 동작이 일치했다고 한다.

예상할 수 있듯이 발표 후 많은 논쟁이 있었다. 장난하나싶어 노여웠던 사람도 있었던 것 같고, 신기하고 흥미로워 하는 사람도 있었던 것 같다. 나는 다른것보다도 쏟아지는 모든 질문에 적절히 방어하는 Rinard 교수의 언변과 설득능력이 신기했다. 프로그램이 이상하게 돌면서 보안 허점이 생길 수 있지 않겠냐는 질문에 마틴 교수는, 보안성을 보장하는 매커니즘이 프로그램이 잘 돌아가야만 돌아가게 할 것이 아니라 프로그램 외부적으로 돌아가게 두는 것이 맞다고 생각한다고 답했다(예를 들면, 기계상태를 모니터링하면서 문제가 감지될 경우 처리). 필립 와들러 교수는 미션 크리티컬한 소프트웨어 대해서 이 방법을 적용할 수 있겠냐고 회의적인 반응을 보였고 이어 둘 간의 열띤 논쟁도 이어졌다. 아마 모든 발표중 질문이 가장 많았던 것 같다.

end-user 들로부터의 에러 리포트를 기대하지 않는 프로그램에 이를 적용해보는 건 좋을 것 같다. 논문에는 수려한 문장들이 쓰여있어 영어공부에 도움될 것 같다.

Code Completion with Statistical Language Models

Veselin Raychev, Martin Vechev (ETH Zürich), Eran Yahav (Technion)

이 연구는 API 를 많이쓰는 프로그래밍에서 사용자의 수고를 덜어주는 것이 목적이다. 예를들어, 왼쪽과 같은 프로그램을 오른쪽으로 바꿔준다. (?)로 표시된 부분은 코드의 구멍으로, 자동으로 채워질 부분이다.

```
void exampleMediaRecorder() throws IOException {
 Camera camera = Camera.open();
 camera.setDisplayOrientation(90);
   ? // (H1)
 SurfaceHolder holder = getHolder();
 holder.addCallback(this);
 holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
 MediaRecorder rec = new MediaRecorder();
   ? // (H2)
 rec.setAudioSource(MediaRecorder.AudioSource.MIC);
 rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
 rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
   ? {rec} // (H3)
```

```
void exampleMediaRecorder() throws IOException {
 Camera camera = Camera.open();
 camera.setDisplayOrientation(90);
 camera.unlock();
 SurfaceHolder holder = getHolder();
 holder.addCallback(this);
 holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
 rec = new MediaRecorder();
 rec.setCamera(camera);
 rec.setAudioSource(MediaRecorder.AudioSource.MIC);
 rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
 rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
 rec.setAudioEncoder(1);
 rec.setVideoEncoder(3);
```

즉 사용자가 어떤 API함수들을 순차적으로 써야할지 헷갈릴 때 이를 도와주는 도구를 개발한 것이다.

방법은 매우 간단하다. 저자들은 API를 쓰는데 일정한 패턴이 있다고 보고 이를 학습하여 예측했다. 다음 확률 모델을 사용한다. 만약 단어 m개로 구성된 문장 s가

있다고 할 때($s = w_1 \cdot w_2 \cdot \ldots \cdot w_m$) 이로부터 어떤 단어 wi가 등장할 확률을 그 앞에 등장한 n개의 단어로부터 계산한다.

$$Pr(s) = \prod_{i=1}^{m} Pr(w_i \mid w_{i-n+1} \cdot \ldots \cdot w_{i-1})$$

이를 N-gram 모델이라고 하고, 저자들이 사용한 것은 3-gram 모델이다.
특정 API 함수 호출이 등장할 확률을 위의 모델로 학습한 다음에 그로 부터 구멍에 들어갈 가장 적절한 API호출을 넣으면 된다. 이 때, API 함수 인자나 함수를 호출할 적절한 객체를 유추하기 위해 간단한 포인터 분석 정보도 가미한다.

자바계열 코드들은 많은 라이브러리 함수를 짜깁기하는 식으로 많이 작성되는데, 이런 류의 프로그래밍에 어느정도 도움이 될 수 있는 아주 간단한 방법이라고 생각했다.

Selective Context-Sensitivity Guided by Impact Pre-Analysis

Hakjoo Oh, Wonchan Lee, Kihong Heo, Kwangkeun Yi(SNU), and Hongseok Yang (Oxford Univ.)

우리 연구실에서 나온 논문이다. 분석 수행 시, 함수 호출 문맥(call context)을 구분하는 것을 효과적으로 하는 것에 관한 내용이다. 함수 호출 문맥 구분하는 것은 분석의 정확도에 큰 영향을 미친다. 그런데 모든 문맥을 구분하는 것은 비용이 매우 비싸기도 하고 의미도 없다. 본 논문에서는 본분석보다 부정확하지만 모든 함수 호출 문맥 을 구분하기 용이한 도메인을 고안하여, 그 도메인으로 모든 함수 호출 문맥 을 구분하는 전분석을 수행하고, 전분석 결과로부터 어떤 함수 호출 문맥을 구분해야 효과적으로 주어진 query를 증명할 수 있는지 뽑아낸다. 그렇게 선택된 함수 호출 문맥 만 구분하는 본분석을 수행하여 정확도를 높이는 방식이다.
이 방법의 약점은, 전분석 결과가 보장해주는게 사실 크지 않다는 것이다. 전분석 결과가 보장해주는 것은 어떠한 문맥을 구분하는 분석을 하는게 아주 쓸모 없는 짓은 아니다(결과가 다 top으로 가지는 않는다)는 정도이고, 주어진 query들을 증명할 수 있다는 보장은 해주지 않는다. 하지만 버퍼오버런에 관해서는, 선택된 query들에 집중하여 많은 query들을 선택적으로 증명할 수 있다는 것을 실험적으로 보일 수 있었다.
발표하기가 어려운 논문이었다. 왜냐하면 문제 자체가 분석을 많이 해보지 않은 사람이면 이해하기가 어려운데다가 내용이 굉장히 기술적이기 때문이다. 심지어 motivating example 조차도 아무리 최소화 시켜도 결코 간단치 않았다. 그래서 나는 이 논문이 어떻게 발표되려나 궁금했다.

학주 형은 일단 앞에 절반은 일반적으로 이해하기 쉬운 내용으로 구성하고 뒤에 절반을 기술적인 내용으로 구성했다. 앞에 절반은 이 논문 자체에 대한 설명보다는 이 연구가 어떤 맥락에서 나온 것인지, 우리 연구그룹이 분석기 관련해서 어떤 일을 해왔고 이 일이 갖는 의미를 설명하였다. 그래서 문외한이 이 발표를 들었어도 이 연구가 분석기 정확도를 필요한 부분만 향상시키기 위한 것이라는 것은 이해했을 것이다. 예상대로 앞에 절반은 비교적 청중들의 집중도가 높았던 반면, 뒤에 절반은 집중도가 높지는 않았다.

2012년 PLDI논문 발표때에도 학주형은 이런 방식으로 발표했는데, 기술적인 디테일이 많이 녹아있는 연구결과를 발표할 때 좋은방식인 것 같다.

Compiler Validation via Equivalence Modulo Inputs

Vu Le, Mehrdad Afshari, Zhendong Su (University of California, Davis)

이 논문은 컴파일러 버그 탐지방법에 관한 것이다. 일반적으로 컴파일러 버그는 잡기가 어려운 편인데 컴파일 에러는 컴파일된 프로그램의 오류로서 간접적인 형태로 나타나기 때문이다. 테스트 케이스의 탐색공간이 큰 것도 이유가 되겠고, 무엇보다 올바르게 컴파일 되었다는 것을 확인하는 방법이 어렵다.

저자들은 아주 신선한 아이디어를 제시하고 있다. (전에 PLDI에도 발표된 바 있던 Csmith를 통해서) 랜덤하게 생성된 코드에 대해 특정 입력값들을 결정 지으면, 그 입력값들에 한해서는 같게 동작하게 코드를 바꿀 수가 있다. 달리 말해, 그 입력값으로부터 진행되는 실행경로 상의 코드를 제외한 나머지 부분을 랜덤하게 고치는 것이다. 그렇게 고쳤을 때 동일한 입력값들에 대해 같게 동작하지 않으면 컴파일러 에러다.

이런 방식으로 147여개의 GCC ,LLVM 컴파일러 버그를 새롭게 발견했다. 아이디어가 이해하기 쉬워서 변환 검산(translation validation)을 위한 일반적인 방법으로 유행할 수 있지 않을까 싶다. 이런 생각을 하게 된 연유에 대해서 궁금했는데 물어볼 기회가 마땅치 않았다. 아쉽다.

Large-Scale Configurable Static Analysis

Mayur Naik (Georgia Institute of Technology)

SOAP라는 워크샵에서 열린 Mayur Naik 교수의 키노트이다. 이 분의 연구팀이 주로 관심 갖는 것은, 유연하고 매개화되어 사용자의 요구조건에 쉽게 맞출 수 있

는 분석이다. 이들이 생각하는 분석은 프로그램과 증명해야할 질의(query)를 받아서 요약의 수준을 결정한다. 분석기는 동일한 요약을 쓰기보다는 질의에 따라 각기 다른 요약을 쓴다.

이들의 관심은 가장 비용효율적인 요약(optimal abstraction)을 어떻게 찾느냐는 것이다. 만약 이 비용효율적인 요약으로 증명이 되지 않는다면 그 성질은 증명이 될 수 없다는 것을 사용자에게 이야기 해줄 수 있으면 좋겠는것도 그들의 바람이다. 이러한 바람으로 POPL'11 부터 이번 PLDI 논문 On Abstraction Refinement for Program Analysis in Datalog 까지 총 4편의 논문을 줄곧 써왔는데 모두 같은 주제를 관통한다. 이 발표에서는 4편의 논문에 대해서 일목요연하게 설명했다.

이번에 우리 연구실에서 낸 논문도 비용효율적인 요약을 다루고 있다는 점에서 목표가 같다. 하지만 우리와 달리 이들은 어떤 질의가 증명 불가능한지에 대해서도 이야기하고 싶어한다. 이 팀은 주로 도메인이 간단한 분석(포인터, 클래스 분석 등)을 고려하고 있기 때문에 그게 가능한 것으로 보인다. 기본적으로 프로그램의 대수적 성질을 고려하여 도메인이 무한한 우리 분석기에서는 생각하기 힘들다.

이 연구팀이 어떠한 철학으로 일을 해오고 있는지를 엿볼 수 있는 좋은 발표였다.

3. 느낀점

근사 계산, 확률 프로그래밍 등 비전통적인 방법들에 기반을 둔 컴퓨팅이 차츰 대두되고 있다는 느낌을 받았다. 첫 번째 키노트가 근사 계산에 관한 것이었고, APPROX 워크샵에서는 정적분석의 안전성을 100% 보장하느라 실용성을 잃을게 아니라 확률적으로 보장 (9x%로 안전하다)하는 방법을 모색해보자는 논의도 있었다. 앞서 소개한 에러를 고치지말고 넘어가자는게 메세지인 논문의 발표도 흥했다.

그리고 또 절실히 느낀 것은 영어 실력의 부족이다. 이번 발표논문 중에 안드로이드 앱 개인정보 유출 분석기(FlowDroid)에 관한 논문이 있었다. 나는 개인적으로 진행하는 연구때문에 FlowDroid 코드를 살펴보고 고칠일이 있었는데, 그 과정에서 생긴 의문들을 직접 저자에게 물어보았다. 물어보려는 것이 영어로 하기 좀 까다로운 내용이라 직접 코드를 보여주고, 종이에 써가면서 소통을 해도 의미가 닿지 않았다. 나는 그저 구현디테일만 알고싶었는데 그 친구는 내가 이 일을 하는 의도와 전체적인 그림을 알고 싶어했다. 나중에는 전달이 잘 안되어 오해가 좀 생겼고 약간 언성까지 높아졌는데, 물어본게 후회가 될 지경이었다. 후에 이메일로 물어보겠다고 대충 마무리하고 명함을 받아왔다. 질문내용을 미리 잘 정리하지 못했던 것

은 잘못이지만, 소통이 잘 안되서 자신감을 잃은 게 더 큰 문제였다고 생각하고, 귀국하자마자 전화영어를 등록했다.

4. 에딘버러

에딘버러는 구시가지가 깨끗하게 잘 보존되어 유네스코 문화유산으로 지정될 만하다는 느낌이 들었다. 구시가지 곳곳이 지은지 몇 백년도 더 된 건물들로 가득하다. 추천할 만한 명소 두 곳은 Calton Hill이라는 곳과 Brew Dog라는 곳이다.

Calton Hill에는 구 시가지 전체를 조망할 수 있는 기념탑이 있는데, 꼭대기에서 살펴보는 경관이 훌륭하다. Brew Dog라는 곳은 맥주집인데 Indian Pale Ale이 유명하다. Indian Pale Ale은 영국이 인도를 지배할 당시 인도까지 맥주를 상하지 않게 보내기 위해서 만든 홉을 많이 넣은 맥주이다. 쌉싸름하고 독특한 풍미가 있다. 그 외에도 이곳은 이곳 사장이 실험적으로 만들어낸 다양한 맥주가 있다. 무려 도수가 14도에 육박하는 맥주도 있다.

그리고 Highland Tour라는 여행상품이 유명한데 우리는 그닥 추천하지 않는다. 스코틀랜드 이곳저곳을 버스를 타고 한바퀴 돌면서 자연경관을 보는 프로그램인데, 괴물이 있는 것으로 유명한 네스호를 살펴보는 것이 주된 볼거리이다. 전체적으로 볼거리가 풍성하지 않고, 이곳 저곳을 이동하는데 거의 모든 시간을 소비한다.

하지만 이건 우리의 개인적인 의견이고 만족스러운 분들도 있을 수 있다.

5. 맺으며

활발하고 열띤 학회 분위기 속에서 마음가짐을 새롭게 할 수 있었다. 내 연구분야가 아니더라도 폭넓은 공부를 꾸준히 해야겠다는 자극도 받았다. 우리 연구실에 방문연구를 왔던 Will Klieber, 구글의 하정우 박사님 등 반가운 얼굴들을 본 것도 좋았다. 생산적인 자극을 받을 수 있도록 늘 지원해 주시는 이광근 교수님께 감사드린다. 또한 이번 출장을 여러가지로 많이 도와주신 ROSAEC 행정팀 분들에게도 감사드린다.