ICSE 2017

아르헨티나, 부에노스아이레스

2017년 5월 20일 - 2017년 5월 27일

서울대학교 허기홍

1. 들어가며

마라도나와 메시, 두 축구 영웅과 반도네온 선율에 맞춘 탱고 춤이 유명한 아르헨티나에서 소프트웨어 공학 분야의 최고 학회인 ICSE 가 열렸다. 오랜 기간 열심히 연구한 결과가 채택되는 영광을 안고, 서울에서 미국 텍사스를 거쳐 부에노스아이레스까지 26시간이 넘는 긴 여행길에 올랐다. 처음으로 가보는 남미이자, 처음으로 참석한 소프트웨어공학분야 학회인데다, PL 분야 학회와는 차원이 다른 어마어마한 규모, 수많은 사람들, 쉴새없이 휘몰아치는 수많은 병렬 세션. 기대, 흥분, 신선함이 넘실대는 파도 위를 신나게 헤엄치는 기분이었다. 원래 2002년 ICSE 를 아르헨티나에서 개최하기로 하고 준비를 하다가, 당시 아르헨티나의 불안정한 정치상황 때문에 학회 몇 달전 급히 미국 올랜도로 옮겼다고 한다. 이 때문에 2002년 ICSE 홈페이지나 포스터에는 미처 바꾸지 못한 탱고 배경이 그 흔적으로 남아 있다. 개회식은 이 역사를 소개하는 것을 비롯하여 아르헨티나 과학기술부 장관의 축사까지 이어지는 화기기애애한 분위기였다.

2. 오랜 여정의 끝

Machine-Learning-Guided Selectively Unsound Static Analysis

허기홍, 오학주, 이광근

제법 오랜기간 붙들고 있던 주제였다. 홍콩과기대에 방문 연구를 갔을 때 알게 된, 지금은 구글에 있는 서현민 박사님이 몇 년 전 잠깐 우리 연구실을 방문한 적이 있었다. 우연히 점심 먹으면서 이야기하다 정적분석기 안전성 (soundness) 의 정도를 조절하는 방법은 없냐고 나에게 물은 것이 시작이었다. 당시 나는 학계에서 분석기의 안전성을 포기한다는 것은 사문난적들이나 하는 짓이라는 생각을 무의식적으로 하고 있었던 듯하다. 그래서 더 신선한 충격이었다. 정적분석의 안전성을 해치지 않는 범위에서 정확도를 조절하는 방법은 이론적으로 잘 정리된 것이 많이 있고 (예, context/flow/field-sensitivity) 그런 것들을 더 유연하게 선별적으로 적용하는 방법을 그 전에 연구한 적도 있었다. 하지만 안전성이라…. 늘 그렇

듯이 시작은 막연했고, 과정은 시행착오로 가득했지만, 포기하지 않고 발전시킨 결과, 좋은 결실을 맺을 수 있었다.

이 연구에서 우리는 기계학습을 사용하여 정적분석기의 안전성을 조절하는 방법을 제시하였다. 실용적인 정적분석기는, 무결점 검증기를 제외하면, 대부분 필연적으로 불안전 (unsound) 하다. 즉, 실제 실행의 모든 경우를 항상 포섭하지는 않는다는 뜻이다. 정확도나 성능 등을 이유로 여러 부분에서 실제 의미의 일부만을 포섭하는 경우가 많다. 예를 들면, 순환문이 한 바퀴만 도는 경우만 고려한다거나, 내용을 모르는 라이브러리 호출은 아무일을 안한다고 가정해버리는 식이다. 그런데 이런 불안전한 기법을 무턱대고 사용할 시에는 분석기 사용자가 알고 싶어하는 정보 (예를 들면, 버그)를 지나치게 많이 놓치는 문제가 있다. 이를 해결하기 위해 이 연구에서는 안전성을 포기해도 버그를 놓치지 않는 지점, 그러면서도 동시에 정확도는 높일수 있는 지점을 잘 찾는 방법을 제시한다. 핵심은 이런 지점을 기계 학습을 이용해서 찾아내는 것이다. 버그가 있는 프로그램의 여러 지점을 안전성을 포기하면서 분석을 해보고, 그 분석 결과를 학습시키는 방식이다. 그렇게 학습된 분류기 (classifier) 는 새로운 프로그램을 분석할 때 안전성을 포기해도 될 만한 부분을 정확하게 짚어주어 분석기의 정확도를 높이는데 도움을 준다. 우리는 이 기술을 C 프로그램을 대상으로 하는 두 가지 정적 분석기 (버퍼오버런 오류 검출기, 포맷스트링 오류 검출기)에 적용하여 그 성능을 실험하였다. 그 결과, 획일적으로 안전성을 포기하는 기존 분석기에 비해서 획기적으로 오류 검출능력을 높일 수 있었다.

발표는 마지막 날 마지막 세션이어서 사람들의 집중도가 많이 떨어지지 않을까 염려지만, 다행히도 여러 사람들이 관심을 가져주었다. 거기서 오갔던 질문을 정리하면 이렇다.

- Q1: 이 기술을 다른 대상 언어 분석이나 다른 안전성 조절에 적용하려면 어떻게 해야하는가?
- A1: 그 때 핵심은 새로운 대상에 대한 특질 (feature) 을 잘 정의해야하는 것인데, 사실 이건 사람이 잘 해줘야한다. 이를 극복하기 위해서 지금 자동으로 특질을 생성하는 연구를 진행하고 있다.
- Q2: 상용 정적분석기에서 이미 선별적으로 안전성을 조절하고 있지는 않나?
- A2: 몇몇 회사 사람들과 이야기 해 본 바로는 그러한 시도가 있긴한 것 같다. 하지만 여전히 개발자가 직관을 바탕으로 수동 튜닝해야하는 큰 문제가 있다. 그래서 우리는 데이터를 바탕으로 하는 체계적인 튜닝 기법을 제시하고 싶은 것이다.
- Q3: 순환문과 라이브러리 호출을 대상으로 안전성을 조절했는데 왜 이 대상을 선정하였는가?
- A3: 우리 분석기 (C 프로그램, 버퍼오버런 분석) 경우에 이 두 가지가 정확도에 가장 큰 영향을 끼쳤기 때문에 그렇다. 다른 분석기도 비슷하지 않을까?

발표 전, 후로도 몇몇 사람들과 만나서 이야기를 나누었다. 대부분 정적분석과 기계학습을 결합하는 시도 특히 정적분석기를 이용하여 고품질 대량 학습 데이터를 생성할 수 있다는 것에 흥미를 보였다.

몇 달, 몇 년을 머리싸매고 고민하며 풀어낸 문제를 많은 사람들에게 소개하고 좋은 반응을 받는 일은 정말 즐겁다. 특히 긴 박사 과정의 끝을 ICSE 발표로 영광스럽게 마무리할 수 있게되어 기쁘다. 험난한 과정 속에서도 늘 격려와 조언 아끼지 않으시고, 멋진 결과를 이루는데 함께 해 주신 이광근 교수님과 오학주 교수님께 감사드린다.

3. 인간은 기록하는 동물, Homo Scriptus

26시간이 넘는 이동시간의 지루함을 달래기 위해 책을 한 권 들고 갔다. 리처드 도킨스의 "확장된 표현형" 이라는 유명한 책으로서 유전과 진화의 근본을 설명하는 책이다. 이 책의 앞부분을 읽으면서 다시금 고교 시절 배운 내용을 상기하였다. 그것은 라마르크식 진화론 (용불용설) 과 대비되는 다윈식 진화론 (자연선택설) 의 핵심 아이디어, 즉 획득형질은 유전되지 않는다는 것. 그래서 20만년전 우리의 초기 조상들이나 우리나 지능면에서 큰 차이는 없을 것이다. 하지만 우리는 고작 수십, 수백년 전 조상들에 비해 훨씬 발달한 문명에서 많은 현상을 이해하며 살고 있다. 이는 인류의 지식이 인간 몸속에 있는 유전자 (gene) 이 아니라 문화에 있는 유전자를 통해 축적, 유전되기 때문이며, 도킨스는 이를 밈(meme) 이라고 부른다.

문화 유전자인 밈이 전파되는 가장 확실한 경로이자 인류 문화 유전의 핵심 중 하나는 “기록”일것이다. 다른 동물과는 달리 본능적으로 기록에 집착하는 인류의 모습. 소프트웨어 공학학회에서도 여실히 느끼고 돌아왔다. 특히, 소프트웨어 저장소에 담긴 기록을 파헤치고 정리하면서 유용한 정보를 얻어내는 소프트웨어 마이닝 분야 학회인 MSR (Mining Software Repository) 에는 기록에 관한 이야기로 가득했다. 흡사 역사학자, 고고학자들의 모임에 온 듯 하였다.

3.1. 기록을 위한 도구

우선 거대한 소프트웨어 저장소에서 유용한 정보를 효율적으로 저장하고 캐내는데 필요한 도구를 만드는 사람들 무리가 있었다. 올해 MSR 에서 발표된 것은 아니었지만, 여러 발표와 토론을 듣다가 알게된 것 중에 GHTorrent[1] 라는 것이 있다. GitHub 에 있는 모든 공개 저장소에서 일어나는 일들을 실시간으로 관찰해서 데이타베이스에다가 찾기 쉽게 기록하는 도구이다. 어느 저장소에서 누가 어떤 커밋을 했고, 어떤 이슈가 등록되었으며, 어떻게 저장소가 연결되고 진화하는지 (fork, pull request) 를 찾기 쉽게 가

[1] http://ghtorrent.org

공하는 일이다. 또한 이 GHTorrent 의 자매 프로젝트인 TravisTorrent[2] 라는 것도 알게 되었는데, GitHub 에서 많이 쓰는 CI (continuous integration) 도구인 Travis CI 가 GitHub 공개 저장소에서 하는 모든 일을 기록하고 저장한다. CI 는 여러 사람이 개발하고 다양한 라이브러리를 가져다 쓰는 상황에서 내 프로젝트가 충돌없이 잘 컴파일/빌드가 되는지 자동으로 검사해주는 도구로서, Travis CI 는 GitHub 에 있는 수많은 소프트웨어의 빌드 상황에서 벌어지는 일을 잘 가공해서 기록한다.

매년 MSR 에서는 "Mining Challenge" 라는 대회를 열어서 이런 정보를 유용하게 사용할 수 있는 재치를 모집한다. 미리 데이터를 정해 주고 능력껏 유용한 정보를 끄집어 내보라는 것이다. 서너장짜리 논문으로 아이디어를 제출하면 선정된 사람들에게 발표할 기회를 준다. 그리고 발표를 들은 청중들이 투표를 해서 1등에게 시상을 한다. 올해는 TravisTorrent 데이터, 2014년에는 GHTorrent, 2015년은 Stack Overflow 데이터 등등. 내가 참여했다면 어떤 것을 만들었을까? 잠시 생각해봤는데, 각 커밋의 특징을 살펴보고 빌드가 성공할지 실패할지 예측해 볼것 같았다. 참가자들도 대부분 비슷한 생각이었는지 빌드 결과 예측 기술을 제시한 사람들이 여럿 있었다. 하지만 그 중 1등을 한 사람은 놀랍게도, 코드가 아닌 개발자의 커밋 메세지를 감정분석 (sentiment analysis) 하여 빌드 성공 여부를 예측하였다.[3] 통계적으로 "love", "happy" 등 긍정적인 마음가짐으로 보낸 커밋은 빌드가 잘 될 확률이 높고, "dislike", "awful" 등 사악한 마음을 갖고 보낸 커밋은 빌드가 잘 안될 가능성이 높다고 한다. 모든 것은 마음먹기에 달렸다는 만고불변의 진리!

사실 여기서 나온 구체적인 결과보다도, 어떤 제약을 주고 참가자들의 창의력을 극대화시키도록 유도하는 학회 분위기가 더 중요하다고 본다. 한 시대를 풍미한 후 결국에 고전이 되는 예술 분야는 대부분 과도하리만큼 경직된 정형성을 갖는 경우가 많다고 한다. 교향곡 (4악장), 시조(3장 6구), 블루스/로큰롤 (12마디, 정해진 코드진행) 등. 자유로운 상황에서 훌륭한 예술이 나올것 같지만 그렇지 않다고. 이런 형식 속에서 등장하는 대부분은 그저그런, 어디서 본듯한 뻔한 작품일테다. 하지만, 그 제약안에서 최대로 자신을 표현하기 위해 모이는 엄청난 에너지는 시간이 지나면 결국 폭발하여 베토벤이나 비틀즈로 발현되는것 아닐까. 우리 분야 (프로그래밍 언어, 정적 분석) 에서도 비슷한 시도를 해보려면 어떻게 하는 것이 좋을지? 다양한 재치를 한데 응집시키는 것은 물론이고, 흥미를 유발하고 진입장벽을 낮추는데도 좋아보인다.

[2] https://travistorrent.testroots.org

[3] Rodrigo Souza and Bruno Silva. Sentiment Analysis of Travis CI Builds.

3.2. 기록을 하는 사람

한편, 잊혀질뻔한 역사를 정리해서 기록하는 작업을 하는 사람도 있었다. MSR 의 기조연설로, Diomidis Spinellis 라는 분이 유닉스(Unix)의 역사를 정리하는 이야기를 해 주었다.[4] 유닉스의 역사는 컴퓨터의 역사라고 봐도 무방할만큼 컴퓨터의 발전과 궤를 같이하여 발전해왔다. 1970년대에 코드 2500 줄로 시작한 유닉스는 2017년에 27백만줄로 발전하였다. 그 사이 벨 연구소, 버클리 대학교 등 여러 단체들에서 BSD 등 다양한 변종을 만들었다. 안타깝게도 이러한 인류 문화 유산이 한데 잘 정리되어 있지 않은 것이 큰 문제였는데, 발표자는 이를 집대성하는 해서 50년 유닉스 역사를 GitHub 에 잘 정리해 두었다.[5] 저장소에 가서 최근 커밋이 48년 전이라고 적혀있는 파일을 보면 소름이 끼친다. 이 역사를 정리하기 위해서 소스 코드를 발굴해가던 과정 이야기는 정말 재미있다. CD-ROM, 플로피 디스크, 심지어 타자기로 쳐놓은 소스 코드 인쇄본까지 구해서 한 커밋 한 커밋 GitHub 에 집어 넣는 장인의 숨결이 느껴졌다. 심지어는 각 커밋을 누가 제출했는지도 기록해 두었기 때문에 과거 벨 연구소와 버클리에 있던 유명한 사람들이 짠 과거 코드도 볼수 있다. 대표적으로 벨 연구소에서 일하던 에릭 슈미트, 현 알파벳 회장.

이 유닉스계의 조선왕조실록 위에서 진행하고 있고, 또 앞으로 진행할 연구 계획도 이야기해주었다. 소프트웨어의 발전 양상, 코딩 스타일 변화, 하드웨어/소프트웨어의 공진화, 각 기관의 프로그래밍 문화, 코드의 수명 등등. 발표에서는 간단한 통계를 몇 개 보여주었다. goto 문의 개수가 지난 60년동안 계속 감소하다가 최근 10년사이에 조금씩 늘고 있었다. 또한 과거에는 모니터가 작아서 그랬는지 변수명, 코드 한 줄이 매우 짧았고 시간이 지날수록 점점 길어지는 양상을 보였다. 또한 이런 방대한 코드를 다루다보니까 평소에는 잘 못느꼈던 git 의 한계점도 많이 느껴서 git 을 개선하려는 일도 계획중인것 같았다.

발표 제목을 보고 그냥 할아버지에게 옛날 이야기 듣는 시간인줄 알고 앉아있다가 눈이 휘둥그레지는 경험을 했다. 몇 년전 우연히 어느 글을 본적이 있다. 기록이 훨씬 쉬워진 디지털 시대의 중요한 역사가 오히려 기록이 잘 안되는 것을 염려하는 사람이었다. 구체적인 사례는 기억이 잘 안나는데, 예를 들어, 현재 인기 있는 웹 사이트들의 초창기 페이지 등 이었던것 같다. 시간이 지날수록, 기술이 발전될수록 디지털 세상이 어떻게 변해가는지, 당시 시대상과 어떻게 연관이 있는지 살펴볼수 있으면 좋지 않을까? 몇 백년후 후손들이 지금 우리의 풍속을 을 엿볼수 있는 기록. 마치 우리가 조선왕조실록이나 김홍도의 그림에서 조상들의 삶을 엿 보듯 말이다. GitHub 가 그런 “디지털 규장각”이 될 수 있을까?

---

[4] Diomidis Spinellis, Half Century of Unix: History, Preservation, and Lessons Learned

[5] https://github.com/dspinellis/unix-history-repo

4. 새로운 버그를 찾아서

정적분석, 버그 탐색 기술 등을 공부하다보니 학회 일정표에서 제일 먼저 눈에 들어오는 것은 당연히 소프트웨어 버그이다. 그동안 나는 주로 프로그램이 잘못된 동작을 해서 죽어버리는 안전성 오류 (safety error), 예를 들면 버퍼오버런, 널-참조, 0으로 나누기 등에만 관심을 가졌다. 헌데, 이번 ICSE 에서는 이보다 까다로운 기능성 오류 (functionality error) 를 대상으로하는 논문이 몇편 있어서 흥미로웠다. 프로그램이 실행중에 죽지는 않지만, 프로그래머가 의도하지 않은 동작을 수행하거나, 입출력은 의도한대로 나오지만 예상과 다르게 성능이 무지하게 느린 경우 등을 대상으로 한다.

4.1. 기록 오류

MSR 에서 밥먹으면서 처음 만났다가 학회 기간 내내 친하게 지냈던 Boyuan 이라는 친구는 프로그램이 엉뚱한 로그를 찍는 부분을 찾는 연구를 ICSE 에 발표하였다.[6] 이 연구에서는 프로그램을 크게 본래 기능을 수행하는 부분 (feature code) 과 과정을 기록하는 부분 (logging code) 으로 나누어 생각한다. 본래 용도를 구현한 부분과 유지보수를 위해 개발자에게 유용한 정보를 제공하는 부분이다. 후자는 보통 특별한 옵션 (예, -g) 을 주었을 때만 실행되는 경우가 많고, 실행이 되더라도 프로그램 동작에는 문제가 없는 경우가 많기 때문에 잘 발견되기 어렵다.

이 연구에서는 수작업을 통해서 흔히 발생하는 6가지 오류 유형을 선정하고 이를 간단히 찾아내는 분석기를 만들었다. 이 오류 유형 중에서는 로그를 출력하는 도중에 널-참조를 하거나 잘못된 타입 캐스팅을 하여 프로그램이 죽는 경우도 있고, 로그의 레벨을 잘못 설정하거나 사람이 읽을 수 없는 내용을 출력하는 것과 같은 경우도 있다. 여기서 찾아낸 버그를 실제 개발자들에게 보고한 결과도 이야기 해주었다. 제출한 버그의 70% 정도는 개발자들이 고쳤고, 10% 정도는 안고치겠다고 연락이 왔다고 한다. 나머지는 아직 논의중이라고. 고친 개발자들 중에서는 “누구도 별 관심 없던 부분에서 버그를 찾아주어 고맙다” 라는 평을 한 사람도 있었고, 거절한 개발자 중에서는 “제발 정적 분석기 결과를 버그라고 제출하지 말아달라” 라는 말도 들었다고 한다. 가혹한 사람이 아닐수 없다.

그동안 생각해보지 못한 문제라서 신선했다. 현재는 단순히 몇가지 정의된 패턴만 검사해서 결론을 내리는 문제와 관련해서 발표 이후에 여러 이야기를 나누었다. 기계 학습을 사용해서 좀 더 일반화 시켜보는건 어떠냐고 물었고 내가 기계 학습을 이용하고 있는 일에 관해 이야기 해주었다.

[6] Boyuan Chen and Zhen Ming Jack Jiang , Characterizing and Detecting Anti-patterns in the Logging Code. ICSE’17

4.2. 성능 오류

한편, 성능 오류 (performance error) 에 관한 논문이 두 개나 발표되어 흥미를 갖고 들어보았다. 과거에 분석기의 성능 오류를 잡기위해 노력하던 와중에 이를 자동으로 탐지하려면 어떻게 해야할지 곱씹어본 적이 있었다. 소프트웨어의 성능을 좌지우지하는 것은 결국 순환문이기 때문에 순환문이 동작하는 양상을 관찰해서 예상치 못한 부분이 나오면 의심해볼만하지 않을까? 당시 나는 출력을 몽땅 파일에 써놓고 수동으로 그런 양상을 관찰했는데, 이 두 논문은 그런 것을 자동으로 하는 연구라고 할 수 있다.

첫번째는 비효율적인 순환문의 흔한 패턴을 나열하고 이를 찾아내는 분석기를 만든 연구였다.[7] 이 논문에서 다루는 패턴은 아래와 같다.

- Resultless loops: 대부분 시간을 결과값에 직접적으로 상관없는 계산을 하는데 쓰는 순환문. 다음과 같은 세부 패턴으로 나뉜다. 아래에 나오는 정규식은 순환문의 동작을 요약한 것인데 0은 아무일도 안함, 1은 어떤 일 (side effect) 을 함을 나타낸다.
  - 0*: 즉, 아무일도 안 하면서 계속 도는 순환문. 이런 것은 삭제해야 한다.
  - 0*1: 즉, 마지막 바퀴에만 일을 하는 순환문. 대표적으로 검색. 이런 경우는 좀 더 빠른 자료구조를 사용해야한다. 예를 들면, 리스트 대신 해시 테이블.
  - [0|1]*: 즉, 돌때마다 동작이 달라지는 순환문. 예를 들어, 어떤 조건이 만족될때만 특정 일을 하는 순환문. 이런 경우는 (가능하다면) 그 조건을 만족 할때만 순환문을 돌도록 바꾸어야 한다.

    for(*){ if(c) /* do something */ } => if(c) { for (*) { /* do something */ } }
  - 1*: 즉, 거의 매번 일을 하는 순환문. 이런 경우는 고치는 방법이 그때 그때 다르다.
- Redundant loops: 이미 했던 일을 여러번 반복하는 순환문. 다시 세부 패턴으로 나뉜다.
  - Cross-iteration redundancy: 이전 순환에서 했던 일을 반복하는 경우. 했던 일은 기억해놓고 재사용함으로써 해결해야한다.
  - Cross-loop redundancy: 다른 순환문에서 했던 일을 반복하는 경우. 위와 마찬가지로 했던 일은 기억해놓고 재사용함으로써 해결해야 한다.

저자들은 각 패턴을 찾아내는 분석기를 간단한 정적분석과 동적분석을 결합하여 만들었다. 정적분석으로 어느 부분이 입력이고, 어느 부분이 결과와 관련된 부분인지 알아낸 다음, 동적 분석을 통해 그 부분이 어떻게 변하는지를 관찰하여 미리 정리한 유형중 어디에 속하는지를 알아낸다.

기술적으로 대단한 부분은 없었지만, 사람들이 흔히 하는 실수 유형을 정리한 점, 대표적인 유형만 쉽게 찾아내는 방법을 만든점이 재미있는 연구였다. ICSE 에는 이런 스타일 연구 논문이 많

[7] Linhai Song and Shan Lu. Performance Diagnosis for Inefficient Loops. ICSE'17

았다. 어떻게 했는지보다는 무엇을 했는지, 결과가 어땠는지에 중점을 두는 분위기인 듯하다. 심지어는 자동화된 방법을 제시하지 않고 사람이 관찰한 결과만 담긴 논문도 많았다.

두 번째 논문도 순환문에 관한 이야기인데 여기서는 특히 자료구조를 순환하는데 집중했다.[8] 자료구조에 접근하는 부분마다 어떤 일을 하는지 기록하게끔 프로그램을 변형시킨 다음 실행을 시키고 기록물을 관찰하는 간단한 아이디어이다. 이 기록물을 보면 프로그램이 자료구조에 어떻게 접근하는지 개발자가 이해할 수 있고, 의도와 다르다면 의심되는 부분을 쉽게 찾을수도 있다. 이렇게 했을때 당연히 실제 순환을 놓치는 경우도 있고, 순환이 아닌데도 순환이라고 여기는 문제도 있다. 이를 해결하기 위한 여러가지 휴리스틱도 같이 제시하였다.

이 논문도 마찬가지로 기술적으로 대단한 부분은 적었지만 결과가 괜찮은 점을 높이 평가받은 듯 하다. 실제 오픈소스 자바스크립트 프로그램을 이런 식으로 분석해서 성능상 오류를 찾아냈고 개발자들에게 보고하여 수정을 이끌어 냈다.

4.3. 전력 소모

신기한 연구주제이다. ICSE 의 포스터 세션에는 전력 소모가 적은 소프트웨어를 만드는 연구를 하는 사람들이 두 팀이나 있었다.[9] [10] 소스 코드 내에서 어떤 부분이 전력 소모가 많은지 찾아내고, 어떤 자료구조, 라이브러리를 사용해야 전력 소모가 적은지 추천해주는 도구를 만드는 사람들이다. 소프트웨어를 잘 작성해서 얼마나 전력 소모를 줄일수 있는지 의아했다. 두 팀의 실험에 따르면 상황에 따라 2%에서 18%까지 줄일수 있다고 한다.

자료구조의 다양한 구현체 중 어떤 것이 에너지 소모가 적은지 미리 테스트 해보고 그 결과에 따라 추천해주는 시스템이다. 주로, List, Set, Map 등 Java 프로그램에서 자주 쓰는 자료구조를 대상으로 했다. 현재는 주로 프로파일링을 통해서 전력 소모가 심한 지점을 찾아내는것 같다. 물어보니 하드웨어의 전력 소비 모델이 있어서 이를 바탕으로 정적분석도 고려해보고 있다고 한다.

최근 5년 정도 사이에 이 두 팀을 중심으로 태동하고 있는 연구 분야인것 같다. 개발자가 짠 코드 뿐만 아니라 컴파일러의 최적화, 하드웨어 설계에 따라 전력소모가 많이 달라질텐데 잘 동작할지는 모르겠다. 하지만 모바일 기기의 사용 시간을 늘이고, 데이터 센터에서 수백억씩 지불하는 전기 요금을 줄이는데 필요한 기술인 것은 분명해 보인다.

---

[8] Rohan Padhye and Koushik Sen . TRAVIOLI: A Dynamic Analysis for Detecting Data-Structure Traversals. ICSE'17

[9] Rui Pereira, Tiago Carcao, Marco Couto, Jacome Cunha, Joao Fernandes and Joao Saraiva. Helping Programmers Improve the Energy Efficiency of Source Code

[10] Jose Benito Fernandes De Araujo Neto, Gustavo Pinto and Fernando Castor. Assisting Non-Specialist Developers to Build Energy-Efficient Software

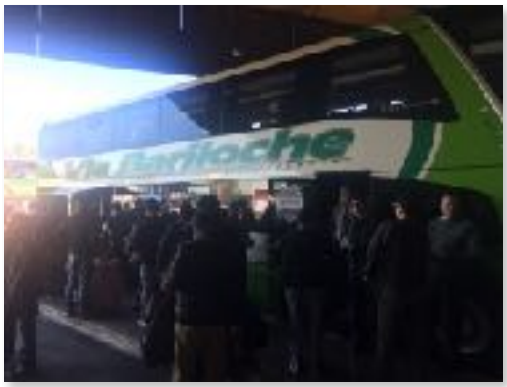
5. 아르헨티나 즐기기

미세먼지가 가득한 서울을 잠시 떠나는 것만으로도 즐거웠던 시간이다. 부에노스아이레스가 전 세계적으로 봤을 때 공기가 깨끗한 편에 속하는 것은 아니었지만 서울에 비하면 상쾌한 곳이었다. 비가 조금씩 오는 날도 있었지만 맑은 날 하늘은 정말 하늘색 물감을 칠해 놓은 것 같고, 흰 구름이 간간이 수놓여 있었으며 햇빛은 쨍쨍했다. 아르헨티나 국기는 분명 저 하늘을 보고 그린 것이리라.

오랜 기간 스페인의 식민지여서 그런지 부에노스아이레스는 여느 유럽도시와 다를 바가 없었다. 사람들의 생김새, 건물, 음식 등이 유럽풍이었다. 스페인은 안가봐서 어떤지 모르겠지만, 호텔을 제외하고는 시내에서 영어가 거의 안통했다. 구글 번역기가 큰 도움이 되었다. 또한, 소고기가 아주 싸다. 덕분에 매일 저녁은 소고기를 즐겼다. 안심, 등심, 갈비 뿐만 아니라 곱창이나 콩팥 구이, 초리소 (스페인식 소세지) 도 적은 돈으로 먹고 싶은대로 사먹을 수 있는 곳이다.

탱고는 아르헨티나의 국민 춤이다. 유명한 탱고 공연장에 들렀는데, 탱고의 영혼이라고 하는 악기, 반도네온의 묘한 선율 위에 흐르는 남녀 무용수들의 춤사위가 일품이었다. ICSE 학회의 리셉션에서는 탱고 댄서들이 와서 참가자들 중 몇명에게 탱고를 가르쳐주는 시간도 있었다. 꼭 제대로된 탱고 극장이 아니더라도 여기저기에서 볼 수 있다. 부에노스아이레스 변두리를 혼자 돌아다니던 날이었다. 어떤 식당 종업원이 아주 기계적인 한국말로 “안녕하세요. 들어가세요. 탱고 보고 가세요.” 하는 말에 이끌려 들어간 작은 식당에서도 탱고를 추는 사람들이 있었다. 그리고 반도네온을 연주하는 할아버지와 그에 맞추어 노래부르는 또 다른 할아버지도 한데 어울린 즐거운 분위기였다.

아르헨티나에 가면 꼭 가보고 싶었던 곳이 이과수 폭포이다. 부에노스아이레스에서 비행기를 타면 두 시간 정도 걸리는데 가격이 너무 비싸서 버스를 이용했다. 자그마치 편도 17시간이 걸린다. 점심을 먹고 버스에 타서 다음날 아침에 이과수에 도착하고, 거기서 한나절 놀다가 다시 저녁에 버스를 타면 그 다음날 점심에 부에노스아이레스에 도착한다. 한반도에서는 상상하기 힘든 코스이다. 생에 버스를 가장 오래 타본 기억은 한창 금강산 관광이 활발하던 시절인 고등학생 때 경남 진주에서 금강산까지 갔던 10시간 남짓인것 같다. 17시간은 눈앞이 깜깜했지만 의외로 견딜만 했다. 우리나라 우등고속 버스보다 좀 더 크고 안락한 좌석이 있는 2층 버스에다가, 화장실이 차안에 설치되어 있고, 비행기처럼 승무원이 밥도 챙겨준다. 버스는 17시간을 쉬지 않고 달린다. 아마 기사 두세명이 교대로 운전을 하는 듯 하다. 버스에는 우리나라 우등/일반같은 등급이 서너 단계가 있다. 1등급과 2등급 버스는 같은 종류 좌석인데 1등급에만 전원과 와이파이가 제공된다. 시간이 안맞아서 2등급을 탔다. 전기도 없고 와이파이도 없이 17시간을 어떻게 버티나 했는데, 많은 생각도 하고, 주는대로 밥도 먹고, 잠도 푹 자다가 일어났더니 금방 도착해 있었다.



이과수 폭포는 정말 너무나 어마어마해서 말로 설명할 수가 없다. 브라질과 아르헨티나의 경계에 있어서 시간이 많은 여행객들은 두 나라에서 보는 광경을 모두 즐기고 돌아간다. 나는 보트투어를 신청해서 폭포 아래쪽으로 가서 물을 흠뻑 맞고, 다시 작은 기차를 타고 꼭대기로 올라가서 이른바 "악마의 목구멍" 을 구경하였다. 이과수 국립공원 입구에 걸린 사진이나, 관광책자, 인터넷 검색 자료 등에는 물이 새하얀 색으로 나와있던데, 뽀샵인듯하다. 실제로 본 폭포는 흙탕물같은 갈색이라 더욱 장엄해 보인다. 너무나 비현실적이어서 컴퓨터 그래픽을 보는듯 했다. 졸렬한 문장에 미처 다 담지 못한 장관은 웹에 공유한 사진으로 대신하려한다.[11]

6. 다시 일상으로

내 생에 가장 긴 여정, 영광스러운 학회 발표, 새로운 분야, 신선한 자극이었다. 오가는 긴 시간 동안, 지난 학위 과정을 다시금 곱씹어 볼 수 있는 좋은 계기이기도 했다. 완전히 새로운 사람들 틈에 홀로 부딪히며 나를 소개해본 것도 좋은 경험이었고 큰 자신감이 생겼다. 늘 큰 꿈을 꾸도록 지도해주신 이광근 교수님과 좌절하지 않게 격려해주신 오학주 교수님께 감사드리면서 이 글을 마친다.

[11] https://goo.gl/photos/z4F1WNZFXPQ6Z4HQ8