ICFP'2010

Baltimore, US, Sep. 27th - Oct. 2nd

소프트웨어 무결점 연구센터 이원찬

요약

미국 볼티모어에서 열렸던 ICFP 2010에 다녀왔다. 첫 국외 학회를 혼자 다녀온 만큼 체험이 조금 남달랐다. 여행기를 통해 보고 느끼고 배운 점들을 나누고자 한다. 또한, 혼자 학회를 나가게 될 다른 사람들에게도 도움이 되었으면 좋겠다.

감상 이후에는 견문을 실었다. 전체 논문들에 대한 내 나름의 개괄 이후에는 즐겁게 읽었던 논문들을 실었다. 이번 ICFP 2010에 실린 논문을 읽고자 하는 사람들에게 도움이 되었으면 한다.

여정과 감상

참석학회

크게 ICFP 본 학회와 작은 워크샵, 심포지엄들로 구성되어 있었다. 그 중 Haskell 심포지엄과 CUFP(Commercial Users of Functional Programming) 학회에 참석했다. Haskell 언어가 함수형 언어에 관심을 갖게 된 계기였던 만큼 Haskell 학회 위주로 참석했다.

가장 절실했던 것

학회는 사람을 만나는 장소라는 점을 절실히 느꼈다. 20분 남짓의 발표에서 사실 많은 것을 기대하기 어렵다. 오히려, 세션 사이사이의 휴식시간에 발표자를 중심으로 많은 것들이 일어난다. 자기소개에 이은 질문에서부터 자신의 관심사와의 교류. 그 과정에서 새로운 연구가 시작되기도 한다. (실제로 허충길 박사님은 학회에서 만난 Neel이라는 친구와 새 연구를 시작하기로 했었다.) APLAS때는 절실히 느끼지 못한 부분이었다. APLAS는 한국에서 열렸고 연구실 사람들이 모두 참석한 상황이었기 때문이었을 것이다.

두 가지가 중요하다고 느꼈다. 나를 알리는 것과 남을 아는 것.

학회를 어떻게 준비할 것인가?

가장 좋은 준비는 논문을 내는 것이다. 더 나아가서는 논문이 되는 일이다. 나를 알리는 가장 좋은 방법이다. 이광근 교수님께서 염려하시는 말씀으로 논문을 안내고 가면 겉돌 수 있다고 하셨다. 그만큼 학회는 발표하는 사람들이 주인공이다. 발표하지 않는 상황이라면 내 연구를 확실히 소개할 수 있어야 할 것이다. 다른 학회에 낸 논문이 있다면 15초안에 소개할 수 있게끔 엘리베이터 스피치(elevator speech)를 준비하면 좋겠다. 실제로, 처음 만난 사람들은 모두가 내게 두 가지를 물어보았다. 나의 연구 분야와 발표한 논문이 있는지. 나는 연구 소개는 잘할 수 있었지만, 아직 발표한 논문이 없어 덜 매력적으로 들렸을 것이다.

그 다음으로 어떤 사람이 무슨 내용으로 논문을 발표하는지 알고 가면 좋겠다. 남을 알자는 취지에서다. 이왕이면 발표된 논문을 미리 읽어보고 가길 권한다. ICFP쯤 되는 학회에 논문이 되면 대부분 그 논문을 미리 볼 수 있게 해놓는다. 논문을 읽고 가면 두 가지 측면에서 좋다. 하나는 짧은 발표시간에 하는 발표를 완전히 이해하는 데 도움이 된다. 또 다른 하나는 질문하기가 쉬워진다. 나는 부족하긴 했지만 몇 편의 논문을 읽고 갔었고 덕분에 발표자와의 대화가 수월했다.

연구에 임하는 우리의 자세

나를 알리는 가장 좋은 방법이 논문을 내는 것인 만큼 연구도 그것에 맞게 준비해야겠다는 생각이 들었다. 논문을 내기 위해 연구를 하자는 말이 아니다. 연구에 집중하고 그 과정을 중간마다 사람들에게 알릴 수 있는 형태로 만들어야 한다는 것이다. 좋은 연구를 하는 것도 중요하지만, 사람들에게 내 연구를 알리는 것 또한 절대 소홀해서는 안 된다고 생각한다.

"사람이 일을 하는 것이 아니라 기한이 일을 하는 것이다."라는 말은 내가 좋아하는 말이다. 기한이 될 때까지 일을 미루자는 것이 아니라 구체적인 목표를 가지고 일을 하자는 말이다. 그러니 지금이라도 당장 목표로 하는 학회를 잡고 구체적인 연구 일정을 잡아보는 것은 어떨까? 다음 ICFP는 일본에서 열린다고 하니 한 번 도전해볼 만하다.

그 밖의 것들

1) 당연한 말이지만, 학회에 참석했다면 적극적으로 임해야 한다. 동료와 함께 왔더라도 따로 떨어져 학회 분위기에 녹아들 것을 권한다. 허충길 박사님과 함께 온 MPI 학생들은 비록 한 팀이었지만 함께 있는 것을 거의 못 봤다. 한편, 일본 학생들은 서로 똘똘 뭉쳐 다른 사람들과 대화를 나누는 것을 거의 보지 못했다. 교류가 자연스러운 학회 분위기가 동양권 학생들이 넘어야 할 가장 큰 산이 아닐까 하는 생각이 들었다.

   식사 때도 용기를 내서 모르는 사람들 사이로 비집고 들어가 보는 것이 좋다. 학회 가서 놀랐던 것은 서로 모르는 사이에도 쉽게 옆자리에 앉아 대화를 나누곤 한다는 것이다. 나도 용기를 내서 Galois사에서 온 Sally와 대화를 나누었다. Galois는 Haskell을 가장 활발히 쓰는 회사로 알려졌다. 주요 제품은 Cryptol이라는 언어로 미국 정부의 암호화 알고리즘을 안전하게 기술하고 구현하기 위해 고안되었다. Cryptol의 특이한 점은 표현 가능한 비트 수로 매개화된 자료형을 사용하여 계산된 값이 특정 비트 수를 넘지 않는 것을 보장할 수 있다는 점이었다. Cryptol 프로그램에 사용된 배열에 대해 정적 분석을 사용한다는 얘기를 듣고 자세히 물어보았다. 아쉽게도 그분은 기술 담당이 아니라서 이 부분에 대해서는 자세히 이야기 듣지 못했다. 하지만, 대답해 줄 수 있는 사람들의 메일 주소를 적어주었다. 대화를 나누지 않았다면 얻을 수 없었던 정보였을 것이다.

2) 영어 공부가 중요하다는 말은 아무리 강조해도 지나치지 않은 듯하다. 우리 분야는 필연적으로 외국인들과 교류를 할 일이 많기 때문이다. 유창할 필요는 없다고 생각하지만 내 생각을 정확하게 말할 수는 있어야 한다.

3) 만일, 학회에 혼자 참석하게 되었다면 룸메이트를 구해볼 것을 권한다. Haskell, ML, Coq 메일링 리스트같이 CFP(Call For Paper)가 많이 올라오는 곳에서 룸메이트를 찾을 수 있을 것이다. 내 경우에는 Haskell 메일링 리스트에서 룸메이트를 구할 수 있었다. (룸메이트는 David Terei라는 친구로 GHC에 LLVM backend를 추가한 인물로 유명하다.) 한 방을 같이 쓴 덕분에 David과 많이 친해질 수 있었다. 다음 ICFP에도 만나자, 그리고 좋은 연구주제가 있다면 함께 하자고 약속했다.

4) 만일, 함수형 언어를 사용하는 일자리를 얻고 싶다면 ICFP를 꼭 참석하는 것이 좋다. Ocaml을 가장 실용적으로 쓰는 회사 중 하나인 Jane Street가 이번 학회를 대대적으로 지원했다. 티셔츠도 나누어주고 두 번에 나누어 취업 설명회도 하는 등 좋은 인재 확보에 열성이었다. 다른 후원사인 Erlang solutions와 Galois도 함수형 언어를 사용하는 회사들이었다.

5) 마지막으로 될 수 있으면 우리 분야의 좋은 학회에 많이 참석하는 것이 좋다고 생각한다. 학계가 돌아가는 분위기도 알 수 있고 사람들과 친해질 수 있는 좋은 계기가 될 수 있다. 잘하는 사람들을 보면서 자극도 받을 수 있다.

ICFP'2010에 발표된 논문들

이번 ICFP 학회에는 99편의 논문 중 33편의 논문이 채택되었다. 33%의 채택률(acceptance ratio)로 학회의 인지도에 비해 도전해볼 만한 수치인 것 같다. 33편의 논문을 내 나름대로 분류해본 결과는 다음과 같다.

분석	2편	프로그램 검증	4편
타입 체계 구현	3편	함수 프로그래밍 및 라이브러리	4편
타입 이론	5편	부분 실행	2편
타입을 사용한 보안	2편	Functional Pearl	2편
Bidirectionalization	3편	Experience Report	3편
병렬 프로그래밍	1편	기타	2편

인상적인 것과 느낀 바를 정리하면 다음과 같다.

1) 우선 분석 논문이 많지 않다는 점이 놀라웠다. (2편 중에 한 편은 우리 연구실 사람들이라면 익히 알고 있을 Matt Might의 논문이다.) 아무래도 정적 분석은 주로 안전하지 않은 언어를 대상으로 하기 때문인 것 같다. 또, 강력한 타입 체계에 대한 자부심도 일조한 것 같다. 함수형 언어의 정적 분석에 대해 좋은 논문을 쓰면 학계에 대한 큰 공헌이 될 것 같다. (개인적으로 Haskell의 메모리 사용량에 대한 정적 분석은 시도해볼 만한 주제라고 생각한다.)

2) 두 편의 Functional pearl 논문은 그 자체로도 쉽고 잘 쓴 글이었지만 단순히 pearl 수준에서 머무는 것이 아니라 굉장히 실용적인 것들을 다루고 있었다. 두 편의 내용은 뒤에서 좀 더 설명하도록 하겠다.

3) 타입 체계 구현을 다룬 세 편의 논문이 인상적이었다. 하나는 타입이 없는 언어인 Racket (구 PLT Scheme)에서 타입을 유추하는 시스템이다. 다른 하나는 Haskell의 타입 클래스 사용 시 문제가 되는 겹치는 타입 클래스 구현 (overlapping type class instance)와 관련된 것이다. 이 둘은 뒤에서 좀 더 설명하도록 하겠다.
나머지 하나는 semantic subtyping에 관한 것이었다. 자료형의 생김새만으로는 하위 타입인지를 검사할 수 없는 경우 SMT solver를 사용하는 것이 핵심 아이디어다. 예를 들어, 기본 타입이 int, float, string밖에 없는 경우 x : !{f:int} && !{f:float}는 x : {f:string}와 동등하지만, 생김새만으로는 이를 알 수 없다. 이럴 때 !{f:int} && !{f:float}가 {f:string}을 의미(imply)하는 것으로 하위 타입 관계를 검사하겠다는 것이다. 이 semantic subtyping은 구현되어 MS의 dminor라고 하는 자료처리 언어에 사용되고 있다고 한다.

4) Bidirectionalization과 관련된 논문이 세 편으로 꽤 활발하다는 인상을 받았다. Bidirectionalization은 내가 이해하기로 주어진 함수의 역함수를 구하는 방법을 말한다. 세 편의 논문 중 Benjamin Pierce의 그룹에서 한 Boomerang 프로젝트가 인상적이었다. Boomerang은 문서 처리를 위한 DSL이다. 주어진 문서를 뷰를 통해 갱신할 때에도 원래 문서의 형태가 깨지지 않게 하는 것이 목적이다. 뷰에서 재배열이 일어나는 경우가 문제가 되는데, 이를 원래 문서와 뷰의 사상 관계를 관리하는 렌즈(lens)로 해결한다. 실용적인 목적을 잘 기술하고 있어 같은 세션의 다른 논문들보다 인상적이었다.

5) 많은 수의 논문이 Coq, Isabelle과 같은 theorem prover로 자신의 정리들을 증명했음을 보였다. 갈수록 더욱 보편화 될 것이기 때문에 공부할 필요가 있다는 생각이 들었다.

6) 타입 이론에 대한 발표는 배경지식이 일천하여 제대로 이해한 발표가 없었다. 발표장에 앉아 있는 내내 부끄럽기 짝이 없었다. 모든 분야를 다 잘 아는 것은 물론 힘든 일이지만 적어도 알아들을 수는 있어야겠다고 생각했다.

두 편의 functional pearl 논문들

"Functional Pearl: Every Bit Counts"는 가장 재밌게 읽었던 논문이다. 타입이 있는 람다 프로그램을 부호화하고 복호화하는 라이브러리에 관한 것이다. 게임을 사용한다는 점이 재밌는데, 게임은 프로그램에 대한 예/아니오로 답할 수 있는 질문으로 구성된다. 부호화는 질문을 반복함으로써 예, 아니오의 리스트를 얻어내는 과정이다. 반대로 복호화는 질문의 답으로부터 프로그램을 되살려내는 과정이 된다. 프로그램 serialization, 코드 손상 검사를 위한 체크섬 등에 사용할 수 있는 매우 실용적인 아이디어다. 논문의 아름다운 점은 예/아니오 질문 게임을 만들기 위한 일반적인 라이브러리를 구축했다는 점이다. 라이브러리는 간단한 게임들로 복잡한 게임을 만들 수 있는 조합기(combinator)들로 구성되어있다.

한 가지 걱정되었던 점은 예/아니오로 대답할 수 있는 질문만 지원한다는 점이었다. 예/아니오 질문을 연속해서 할 때 게임을 표현하는 트리는 이진 트리의 형태가 된다. 따라서, 질문을 많이 반복한다면 메모리 사용량이나 처리 속도 측면에서 문제가 있을 것 같았다. 저자인 Dimitrios에게 이를 물었으나 메모리 사용량은 염려되는 부분이지만 처리 속도는 크게 신경 쓰지 않는 것 같았다. 그의 말로는 질문 하나하나의 처리가 매우 간단하여서 최적화될 것이라고 했다. 실제 사용되는 언어들에 적용해볼 예정이라고 하는데 그 결과가 기대된다.

두 번째 functional pearl인 "A Play on Regular Expressions"는 Haskell로 구현한 정규 표현식 라이브러리에 관한 것이다. 기술적인 내용에는 특별히 놀라운 점은 없었다. 하지만, Haskell을 사용해서 고성능 정규 표현식 라이브러리를 만들 수 있었다는 점이 놀라웠다. (Google에서 최근에 만든 C로 된 정규 표현식 라이브러리보다 빠르게 동작했다고 한다.) 매치되는 문자열을 모으는 기능과 다른 문자열로의 치환 기능이 추가되면 정말 쓸만해 질 것 같다. 또 다른 놀라운 점은 논문이 희곡 형태로 구성되어 있다는 점이다. 논문을 Haskell 전문가, 추상화(abstraction) 전문가, 오토마타 전문가 셋이서 라이브러리를 완성해가는 과정에 나눈 대화로 그리고 있다. 형식에 구애받지 않고 사람들이 쉽게 접할 수 있게 만든 점이 훌륭했다. 그 덕분에 사람들이 매우 좋아했던 논문이었다. 개인적으로도 가장 pearl다운 pearl이 아니었나 하고 생각한다.

"Instance Chains: Type Class Programming without Overlapping Instances"

타입 클래스를 쓸 때 문제가 되는 겹치는 구현(overlapping instances) 문제를 해결하기 위한 한 가지 시도를 다루고 있다.

겹치는 구현 문제는 다음과 같다. 타입 클래스는 타입 혹은 타입의 관계에 대한 집합으로 볼 수 있다. 타입들 간의 관계를 새로이 집합에 가담시키는 방법은 구현을 추가하는 것이다. 문제는 서로 다른 구현이 하나의 타입에 동시에 해당할 수 있다는 점이다. 예를 들어, C라고 하는 타입 클래스가 있고 ([a], b)와 (a, [b])에 해당하는 구현이 있다고 하자. ([a]는 a의 리스트를 말한다.) 만일, ([a], [b])에 대한 구현을 찾을 때 ([a], b), (a, [b])가 동시에 해당된다. Haskell의 타입 클래스 구현이 나중에 추가될 수 있다는 사실은 문제를 더욱 악화시킨다. 왜냐하면, 처음에는 ([a], b)에 대한 구현만 있다가도 나중에 다른 사용자가 (a, [b])에 해당하는 구현을 추가할 수 있기 때문이다.

겹치는 구현이 존재할 때에 GHC는 주어진 타입에 더 가까운 구현을 선택한다. 가까운 구현이라 함은 그 구현의 타입에서 주어진 타입에 이르기까지 필요한 타입 변수의 치환 횟수를 말한다. 예를 들어, [Int]는 [a]보다 [Int]와 더욱 가깝다. 하지만, 앞에서 설명한 경우는 어느 한 쪽도 다른 쪽보다 더 가깝다고 말할 수 없다.

논문은 해결책으로 적용할 구현 사이에 고정된 순서를 두는 방법을 제안한다. 즉, 주어진 타입이 ([a], b)로 표현될 수 있는지 먼저 살펴보고 그다음 (a, [b])를 적용하는 것이다. 이를 위해 겹칠 수 있는 구현은 한꺼번에 선언할 수 있게만 제한한다. 이 덕분에 다른 곳에서 추가된 구현은 다른 어떤 구현과도 겹치지 않는 다는 것을 보장할 수 있다. 이 타입 클래스 체계가 안전함을 증명하기 위해 타입 클래스의 의미를 엄밀히 정의한 것이 논문의 핵심이다.

“Logical Types for Untyped Languages”

타입이 없는 Racket언어에서 타입을 유추하기 위한 타입 체계의 구현을 다룬 논문이다. 타입을 정적으로 알아내기 위해서 조건문에서 사용하는 타입 검사 구문에서 정보를 얻는다. 예를 들어, (cond [(number? x) (body)] ...)와 같은 프로그램에서 body에서의 x의 타입은 숫자라는 것을 확실히 알 수 있다. 이 같은 조건문마다 만족 될 경우와 아닌 경우에서 타입을 모아 각 문장의 타입을 결정한다. (이와 같은 타입 체계를 occurrence typing이라고 부른다고 한다.) 이 아이디어를 가져다가 많이 쓰이는 Python이나 Ruby에 적용해보면 재밌을 것 같다는 생각이 들었다.

저자 중 한 명인 Matthias Felleisen은 프로그래밍을 가르치는 일에도 열심인 분이다. 이 논문에서도 그의 노력을 엿볼 수 있었다. 타입이 없는 언어를 잘 만들어서 아이들에게도 가르칠 수 있게 한다. 다른 한편으로는 타입 체계를 지원함으로써 안전한 구현도 가능하게 만든다. 두 가지를 모두 잘할 수 있는 언어로 Racket을 발전시키고 있구나 하는 생각이 들었다.

Haskell 심포지엄과 CUFP

“Hoopl: A Modular, Reusable Library for Dataflow Analysis and Transformation”

사실 이 논문은 발표되기 훨씬 전에 읽어본 논문이다. (Hoopl을 사용하여 “구간 도메인 분석기, 5분 안에 만들기”라는 튜토리얼 자료를 만들기 위해서였다.) 데이터 흐름 분석을 위한 일반적인 프레임워크로 그래프 형태의 중간 언어를 대상으로 하고 있다. 지난 POPL과 ICFP 모두에서 떨어지고 우여곡절 끝에 이번 Haskell 심포지엄에 채택된 비운의 논문이다.

Hoopl은 2002년 POPL논문인 “Composing dataflow anlayses and transformations”을 구현하고 있다. 핵심은 데이터 흐름 분석의 결과로 코드를 최적화하고, 최적화된 코드에 분석을 다시 수행하는 것을 반복하는 것이다. 사용자로 하여금 분석과 코드 재작성을 잘 기술할 수 있게끔 필요한 부분에만 빈칸을 만들어두었다.

Haskell 논문답게 타입 체계의 위력을 곳곳에서 발휘하고 있다. 가장 눈에 띄는 것은 타입을 통해 잘못된 형태의 코드를 만들지 못하게 했다는 점이다. GADT를 사용해서 각 블록이 부모, 자식 블록에 대해 열려 있는지를 타입 정보로 부호화했다. 예를 들면, 함수를 시작하는 블록은 부모에 대해서는 닫혀 있고 자식에 대해서는 열려 있는 타 입을 가진다. 반대로 함수를 끝마치는 블록은 자식에 대해 닫혀 있고 부모에 대해서는 열려있다. 그다음, 서로 열려 있는 블록들 간의 연결만 유효하도록 연결 함수의 타입에 제한을 두었다. 즉, 자식에 대해 열려 있는 블록과 부모에 대해 열려있는 블록만 연결할 수 있다. 이같이 형태에 대한 제약을 타입에 부호화하여 정적으로 프로그램 그래프의 생김새를 검사할 수 있게 된다.

Simon Peyton Jones가 직접 발표해서 더욱 기억에 남는다. Haskell을 만든 사람이니 심포지엄에 오겠거니 했지만 직접 발표할 줄은 꿈에도 몰랐다. Hoopl을 구성하는 각 모듈의 타입만 가지고 왜 타입이 그러해야만 하는가를 일목요연하게 설명해나갔다. 크리스티나가 왜 Simon Peyton Jones의 발표가 인상 깊었다고 했는지 알 것 같았다.

High performance Haskell

내가 이번 학회에서 가장 기대했던 세션이었다. 산학 과제에 Haskell을 적용하면서 겪었던 문제들의 해답을 찾고 싶었다. 하지만, 대부분 내가 알고 있는 사실들을 재확인했던 자리라 조금은 실망스러웠다. 고성능 Haskell 코드를 작성하기 위한 지침을 정리하면 다음과 같다:

1) 될 수 있으면 적극적 계산(strict evaluation)을 활용하라. Haskell은 지연 계산(lazy evaluation)을 하는 언어이다. 모든 계산은 필요하기 전까지 포장된 thunk 형태로 보관된다. 따라서, 당장 결과가 필요한 경우 이는 순수 비용이 된다. (아이러니하게도 대부분 결과가 당장 필요하다.) 따라서, 다음의 두 가지 방법을 통해 적극적 계산을 유도하는 것이 필요하다. 첫 번째는 추상 자료형(abstract data type)에 적극적으로 계산하는 필드를 두는 것이다. 적극적으로 계산되는 필드는 자료형 앞에 느낌표가 붙어 있다. 두 번째는 함수 인자 앞에 느낌표를 붙이는 것이다. (이를 bang pattern이라고 부르는데 사용을 위해서는 GHC의 확장된 기능을 활성화 해야 한다.) 적극적으로 계산되는 필드 혹은 인자에 대해서는 값이 묶이기 전에 계산이 완료되게 된다.

2) Unpack pragma를 사용하라. Haskell의 모든 자료형은 타입 정보와 함께 포장되어 있다. 단일 타입의 적극적으로 계산되는 필드는 타입 정보가 필요 없으므로 떼어버릴 수 있다. 자료형을 떼고 난 다음에는 부모 자료형에 곧바로 내장시킬 수 있게 된다. 예를 들어, Foo Int로 선언한 경우 Foo는 타입정보 저장을 위해 4바이트, Int에 대한 포인터 4바이트를 차지하고, 추가로 Int 자료형 또한 8바이트를 차지한다. (4바이트는 타입정보, 4바이는 실제 정수 값) 한편, Foo !Int의 경우 Int 자료형을 위한 8바이트를 절약할 수 있게 된다.

3) 함수의 각 인자의 계산 여부는 반환되는 값 (return value) 계산에 쓰이는지로 알 수 있다. 반환 값에서부터 역으로 추적하면서 인자가 그 계산에 이바지하는지 확인하면 된다.

4) 끝 재귀호출(tail-recursive)을 사용하라. 다른 함수형 언어와 마찬가지로 Haskell에서도 끝 재귀호출에 대해 최적화를 수행하기 때문이다.

5) Core 언어로 변환된 결과를 참조하라. GHC의 -ddump-simpl 옵션을 사용하면 core로의 변환 결과가 출력된다. Core에서는 case문에서만 계산이 일어나기 때문에 내가 만든 코드의 동작을 더 쉽게 예상할 수 있다. 실제로 많은 Haskell 해커들이 한 화면에는 Haskell 소스를, 다른 화면에는 core를 띄워놓고 작업한다고 한다.

6) 프로파일링을 사용하라. GHC는 다양한 프로파일링 옵션을 지원한다. 시간 및 메모리를 많이 사용하는 함수를 순위별로 출력해준다. 또, 시간에 따른 메모리 변화량을 그래프로 그려주기도 한다. 메모리 사용량은 함수별, 자료형 별로 출력 가능하다. 프로파일링 결과를 통해 문제를 일으키는 함수를 찾는데 도움을 얻을 수 있다. 실제로, Haskell로 프로그램을 만들 때 메모리 사용량이 항상 문제가 되는데, 이때 프로파일링이 도움된다.