APLAS / CPP 2011

December 4-9, 2011, Kenting, Taiwan

서울대학교 프로그래밍 연구실 조성근

학회 참가

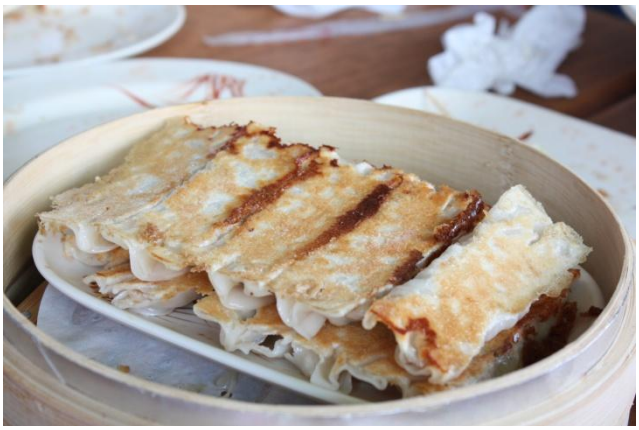
이번 APLAS와 CPP는 대만의 켄팅이라는 곳에서 열렸습니다. 켄팅은 대만의 남쪽 끝에 위치한 바닷가 마을입니다. 이곳은 겨울인데도 날씨가 따뜻하고 맑아서 마치 우리나라의 가을과 같은 분위기였습니다.

CPP는 올해 처음 생긴 학회입니다. Certified Programs and Proofs의 약자로서 프로그램의 검증과 기계화된 증명에 대해 이야기하는 학회입니다. 처음 생긴 학회임에도 저명한 학자들로 초청 강연과 위원회가 구성되어 있어 알찬 학회 프로그램을 경험할 수 있었습니다. 좋은 경험을 할 수 있게 기회를 주신 ROSAEC센터에 감사를 드리며 제가 경험한 이야기를 시작하겠습니다..

카드 결제의 어려움

대만은 신용카드 사용이 제한적이었습니다. 대부분의 지출을 신용카드로 하려고 환전을 최소한으로 하였는데 이것이 실수였습니다. 세븐일레븐을 비롯하여 음식점, 고속버스 티켓 구매가 모두 현금으로 이루어질 수 밖에 없었습니다. 이런 사실을 미리 알았더라면 현금을 조금 더 넉넉하게 가지고 갔을 텐데 그렇지 못 해 당황했습니다.

맛있는 길거리 만두



신용카드를 사용하지 못하여 대만에 대해 상당히 실망하던 가운데 이를 멈추게 한 것이 바로 대만의 맛있는 음식이었습니다. 우리는 Zuoying에 들러서 점심을 먹었습니다. Zuoying의 역 근처는 매우 평화로운 분위기였습니다. 그 중 한 우육탕면 집에 들어갔습니다. 여러 가지 만두와 우육탕면을 시켰는데 모든 음식이 매우 만족스러웠습니다. 지금까지 먹어본 만두 중 맛있기로 순위에 꼽히는 집이었습니다. 그냥 *동네의 만두 집*이 이 정도인데 정말 맛있다고 소문난 집은 어느 정도일까? 하는 생각이 들었습니다.

의사소통의 어려움

우리가 도심이 아닌 시골로 여행을 가서인지 영어가 통하지 않는 가게가 상당히 많았습니다. 유창한 영어까지는 아니어도 물 정도는 요구하고 서비스 받을 수 있을 줄 알았는데 그렇지 못했습니다. 음식점에서 생수를 얻는 데에 여러 친구들의 바디랭귀지가 사용되었습니다. 도심이 아닌 마을로 여행을 떠날 때에는 그러한 기본적인 단어나 음식의 이름들을 미리 알고 떠나는 것이 좋다는 교훈을 얻었습니다.

바다 구경

12월 4일의 오후 튜토리얼이 다른 시간으로 변경되어 바다 구경을 하기로 하였습니다. 호텔이 바로 바다 앞에 있었기 때문에 해변으로 나가는 데에는 오래 걸리지 않았습니다. 바람이 아주 강하게 불었습니다. 한국에서는 태풍이 왔을 때에나 느껴봤을 바람이었습니다. 햇볕은 따스했습니다. 도저히 12월의 날씨라고는 믿기지 않았습니다. 정말 좋은 때에 이곳에 왔다는 생각이 들었습니다. 여름에 왔다면 상당히 더웠을 것입니다.

[TUTORIAL] Dependently Typed Programming in Agda.

Shin-Cheng Mu.

첫 날의 튜토리얼에서는 dependent type에 기반한 증명 보조기 Agda를 소개하였습니다. 튜토리얼을 맡으신 분은 Academia Sinica의 Shin-Cheng Mu 교수님이었고 최근에 locally

nameless를 이용하는 Church-Rosser 정리를 Agda로 증명했다고 합니다. 지금까지 증명 보조기는 Coq밖에 경험하지 못했기 때문에 Agda는 어떨지 궁금했습니다.

Coq과 Agda는 상당히 닮아 있다는 것이 첫 느낌이었습니다. 먼저 두 증명 보조기 모두 대화형(interactive)으로 증명이 진행됩니다. 따라서 증명 중간중간 전제와 결론을 확인할 수 있으며 그에 따라 앞으로의 증명을 어떻게 진행할지 결정하게 됩니다. Dependent type을 이용한 증명에서는 증명내용을 한 번에 생각하기가 쉽지 않으므로 Coq이나 Agda와 같은 증명 보조기가 대화형으로 진행된다는 것은 중요한 점이라고 합니다.

증명이 진행되는 과정도 상당히 비슷한 모습을 보입니다. Coq은 전제와 결론(목표)가 있을 때 tactic이라 불리는 명령들을 이용해서 전제와 결론을 수정해 나갑니다. Agda에서는 전제의 내용을 tactic을 이용해서 직접 수정하는 대신에 전제의 내용을 case별로 나누어 증명식을 만들어 나갑니다. 이는 Coq의 inversion tactic과 유사한 모습을 보입니다. 또한 auto 기능을 이용해서 간단한 경우 증명을 자동으로 만들기도 하는데 이 또한 Coq의 auto tactic과 비슷하다는 느낌을 받았습니다. Coq에서는 사용자가 원하는 대로 이러한 자동화를 디자인할 수 있기 때문에 평균적으로 Coq의 증명이 Agda의 증명보다 짧다고 합니다.

발표의 후반부는 기계화된 증명에서 변수 이름짓기(variable binding)을 다루는데에 나타나는 어려움에 대한 이야기였습니다. 사람이 손으로 하는 증명에서는 보통 nominal 방식의 변수 이름짓기 방법을 사용하는데, 기계화된 증명에서는 어느 변수가 어느 람다식에 의해서 묶이는지, 혹은 자유변수인지를 판단해야 하는 귀찮은 작업들을 수반하게 됩니다. 그래서 도입된 것이 de Bruijn 방식인데 이것의 문제점은 beta-변환에 의해서 변수의 값들이 민감하게 변화한다는 점을 들 수 있습니다. Locally nameless 방식은 이들의 단점들을 어느 정도 보완한 방법이지만 이 방식을 사용하여도 locally closed를 확인하는 많은 증명과정을 피할 수 없다는 점을 이야기하였습니다. 결국 지금까지 제시된 많은 방법들이 모두 문제점을 가지고 있다는 느낌을 받았습니다. 발표자의 증명에서도 많은 부분이 infrastructure를 위한 증명이었다고 하였고, 자동화 또는 반자동화된 infrastructure 생성에 관한 연구의 중요성에 대해 이야기하였습니다.

Access-Based Localization with Bypassing.

Hakjoo Oh and Kwangkeun Yi.

같은 연구실의 오학주 학생의 발표였습니다. 이미 연구실 내부 세미나와 ROSAEC 워크숍에서 여러 번 이야기했던 내용으로서, 프로그램을 분석할 때 분석에 필요한 내용만을 넘겨 줌으로써 분석의 성능을 향상시키는 내용이었습니다. 특히 bypassing은 함수 안에서 다른 함수를 호출할 때 발생하는 비효율적인 요약 메모리 이동을 막아줍니다. 이 방법을

통해서 기존의 분석 중 커다란 함수 호출 사이클을 가지는 프로그램의 분석 속도를 크게 향상시키고 메모리 사용을 줄일 수 있습니다.

학주의 발표는 매우 인상적이었습니다. 문제점이 어떤 것인지, 어쩌다가 그런 문제가 발생했는지, 이를 어떻게 해결했는지, 그 결과는 어떠한지가 모두 정확하게 표현되었습니다. 특히 처음에 현재 분석기의 성능을 소개함으로써 사람들에게 좋은 이미지를 준 것도 발표의 집중도를 올리는 데에 도움을 주었다고 생각합니다.

Polymorphic Multi-stage Language with Control Effects.

Yuichiro Kokaji and Yukiyoshi Kameyama.

Yukiyoshi Kameyama 교수님은 제가 학부 4학년 때 있었던 연구실의 지도 교수입니다. 그것이 2005년이니 벌써 6년이 지났습니다. 그리고 이번에 학회에서 만나게 되어 정말 반가웠습니다. 타국에서 생활하면서 가장 의지가 되었던 일본인이었기에 어떤 일을 하고 있는지 더욱 관심이 가게 되었습니다.

이 연구는 다형 타입, 다단계 언어, 조종 효과(control effect)를 모두 포함하는 언어를 디자인하는 것을 목표로 합니다. 이 세 가지 특징 중 두 가지를 가지고 있는 연구는 몇몇 진행되어 왔으나 세 가지 모두를 가지는 언어는 지금까지 없었다고 합니다. 발표에서는 세 가지를 모두 다룸으로써 생기는 모든 문제를 다루고 있지는 않았고, 주로 다형 타입을 위한 문법 제약에 대해 설명하였습니다. 발표 시간이 길지 않아 전체 내용을 모두 다루지 않은 것이 아쉬움으로 남습니다.

조종 효과(control effect)라는 것은 쉽게 말해 나중에 실행할 것(continuation)을 함수의 인자로 주거나 결과로 출력하는 계산들을 이야기합니다. 이는 Lisp의 catch/throw, ML의 exception, Scheme의 call/cc와 비슷한 개념으로써 프로그램의 실행 흐름을 조종한다는 의미로 이해할 수 있습니다.

이 연구의 핵심은 다형 타입을 위한 let식의 문법을 제한함으로써 세 특징을 모두 포함하는 안전한 언어를 디자인하는 것입니다. 문제는 let식(`let x=e in e'`)에서 `e`를 일반화하기 위해서 상태(context)를 알 필요가 있다는 점입니다. 이를 해결하기 위해 널리 알려진 방법은 `let x=e in e'`에서 `e`를 값으로 제한하는 것이지만 이는 다단계 프로그램의 표현력을 상당히 제한하므로 사용하기에 곤란합니다. 이 연구에서 해결책으로 제안하는 것은 purity 제약입니다. 식 `e`에 값을 사용하거나 reset식인 `{e''}`을 사용하는 방법입니다. 식 `e`가 그 밖의 식인 경우 reset을 추가하여 일반화의 조건을 만족시키도록 변환할 수 있습니다. 이러한 방식의 변환이 기존의 프로그램의 의미를 바꾸지 않는다는 것은 증명되어 있습니다.

발표에서는 타입 규칙들이나 그 안전성에 대해 이야기하지 않았지만 논문은 타입 추론 알고리즘에 대한 간단한 설명까지 포함하고 있습니다. 나중에 시간을 두고 꼼꼼히 논문을 읽어보고 싶다는 생각이 들었습니다.

포스터 세션

APLAS에는 포스터 세션이 있었습니다. 포스터 세션은 발표자와 가까이에서 설명을 듣기 때문에 내용 이해가 더욱 수월하고, 모르는 것도 부담 없이 물어볼 수 있다는 장점을 가지고 있습니다. 이 때문에 ROSAEC 센터의 워크숍에서도 포스터 세션을 도입한 이후에 서로의 연구에 대해 더욱 잘 이해하게 되었던 것으로 기억합니다. 다만, 포스터 세션의 시간이 모든 포스터의 내용을 듣기엔 짧았기 때문에 그 점이 아쉬웠습니다.

[POSTER] Programming with Infinitesimals:

A While-language for Hybrid System Modeling.

Kohei Suenaga and Ichiro Haqsuo.

이 연구의 목적은 연속된 정보(continuous data)와 연속되지 않은 정보(discrete data)를 동시에 다루는 하이브리드 시스템을 표현(modeling)하고 검증하는 데에 있습니다. 이를 위한 방법으로는 대표적으로 연속된 값에 초점을 두는 조종이론(control theory)과 테스트에 기반을 둔 방법이 있습니다. 이 연구는 이들과 다르게 프로그램 검증 기술을 문제 해결에 사용하고 있습니다. 프로그램 검증기술이 프로그램 검증이 아닌 다른 시스템 검증에 사용된 것을 처음 접했기 때문에 신기하였습니다.

핵심 아이디어는 그러한 시스템을 프로그램으로 표현하기 위해서 프로그래밍 언어 While에 매우 작은 값(infinitesimal)을 나타내는 dt를 추가하는 것입니다. dt가 추가된 While은 실제로 실행되기 위한 프로그래밍 언어가 아닌 검증되기 위한 프로그래밍 언어입니다. 하이브리드 시스템을 표현하는 데에 그 의미가 있으며, 따라서 컴파일러 또한 필요하지 않습니다. 새로운 언어를 위한 검증기 또한 dt값을 고려합니다. 기반이 되는 논리는 1960년대에 Abraham Robinson에 의해 만들어진 nonstandard analysis입니다.

[POSTER] Towards Incremental Resource Usage Analysis.

Elvira Albert, Jesús Correas, Germán Puebla, and Guillermo Roman-Duez.

이 연구에서는 기본적으로 프로그램의 자원(시간과 메모리)의 최대이용을 분석합니다. 더욱 집중하고 있는 것은 점진적인(incremental) 분석입니다. 즉, 어떤 프로그램을 분석한 후

프로그램을 수정하였을 때, 분석을 처음부터 다시 하는 것이 아니라 이전의 분석결과를 최대한 이용하려는 것이 연구의 핵심입니다. 이를 달성하기 위하여 연구팀은 분석 결과 사이의 의존성을 기록해 놓습니다. 그리고 그러한 의존성에 의하여 영향을 받는 부분들만 다시 분석한다는 것입니다. 이 연구팀의 분석기는 Java의 바이트코드에서 각 함수의 분석된 결과(summary)를 기록하여 전역분석(global analysis)을 수행하였습니다.

포스터 발표였기 때문에 그 장점을 살려 직접 분석기가 돌아가는 모습을 보여주었습니다. 사실 자원의 사용량 분석이 프로그램의 종료분석과 밀접한 관련이 있고, 종료분석은 끝나요문제(halting problem)이기 때문에 끝나지 않는 프로그램에 대해서는 어떠한 결론이 나오는지 물어보았습니다. 예상대로 종료를 분석하기 어려운 프로그램에 대해서는 자원을 얼마나 사용할지 모른다는 결과를 내어주게 되어 있다고 했고, 이를 직접 실행하여 보여주었습니다. 분석기는 Eclipse의 plugin으로 되어 있었으나 코드를 구할 수는 없고 연구팀의 홈페이지[1]에서 분석기를 돌려볼 수는 있다고 합니다.

[INVITED TALK] Software Verification with Liquid Types.

Ranjit Jhala.

Ranjit Jhala의 발표를 듣는 것은 두 번째입니다. 지난 번에 MIT에서 열렸던 SAT/SMT 여름학교에서 했던 발표에서 그의 발표는 상당히 인상적이었습니다. 그는 발표할 때 매우 열정적으로 청중들과 소통합니다. 마치 연극을 한다는 느낌을 받을 정도입니다. 그의 발표를 또 다시 볼 수 있게 되어서 개인적으로 매우 기뻤습니다. 발표는 liquid type에 관한 것입니다. 이번 발표도 지난 여름학교에서의 발표와 비슷한 내용이었습니다.

Liquid type은 메모리의 안전(buffer overflow, null dereference)을 나타내는 성질을 자동으로 검사하기 위한 타입입니다. 데이터의 구조만을 나타내는 보통의 타입과는 다르게 liquid type에는 값을 제한하는 논리가 포함되어 있습니다. 이러한 논리식의 추론은 다음의 순서로 이루어집니다.

- 프로그램의 각 값에 해당하는 타입에 liquid type의 논리에 해당하는 변수를 기록
- Subtyping을 이용하여 제약식을 만듦
- 제약식을 SMT로 풀어 논리가 포함된 타입을 구함

발표자는 기본 타입, 구조를 가지는 타입(collections), 함수 타입에 대해서 어떻게 liquid type이 구해지는지 예를 보이고, 데이터 타입의 unfold, fold와 liquid type의 instantiation,

[1] http://costa.ls.fi.upm.es/

generalization을 비교하여 설명하였습니다. 웹페이지[2]에서 liquid type을 경험해 볼 수 있습니다. 발표자가 여러 성공적인 실험 결과를 보여주며 보여주었던 이야기가 기억에 남습니다.

"분석이 더 똑똑해질수록 왜 분석이 실패했는지를 보여주는 것이 더 어려워진다."

A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses.

Michael Armand, Germain Faure, Benjamin Gregoire, Chantal Keller, Laurent Thery, and Benjamin Werner.

이번 CPP에서 적지 않은 논문들이 "어떻게 SAT/SMT solver를 Coq과 같은 증명보조기에서 활용할 수 있을까?"에 대한 해답을 주었습니다. 이 논문도 그러한 논문 중 하나였습니다.

SAT/SMT solver를 증명보조기에서 사용하는 경우는 크게 다음과 같이 두 가지가 있습니다.

- 자립적 방법(autarkic approach): 증명보조기 내부에서 안전성이 검증된 solver를 직접 만들어 사용하는 방법입니다.
- 의심 많은 방법(skeptical approach): 외부의 증명보조기를 사용하고 그로부터 증명에 대한 힌트(proof witness)를 받아 증명을 다시 구축하는 방법입니다. 일반적으로 이 방법이 위에서 말한 자립적 방법보다 검증할 것이 적습니다. 이 논문에서 이야기하는 방법도 여기에 속합니다.

증명에 대한 힌트(proof witness)

SAT/SMT solver가 주어진 질의(query)에 대한 예/아니오 대답뿐 아니라 증명을 위한 힌트를 함께 내어주게 됩니다. 그러한 힌트를 받은 Coq은 SAT/SMT solver가 했던 내부적인 증명을 새롭게 구축하게 됩니다. Coq은 증명의 힌트를 받을 뿐이지 SAT/SMT solver의 결과를 그대로 사용하지는 않으므로 증명의 안전성(soundness)는 그대로 유지됩니다. 증명에 대한 힌트는 SAT/SMT solver의 resolution chain입니다. SAT/SMT solver에서의 resolution을 그대로 따라가며 증명을 구축하는 것입니다.

[2] http://goto.ucsd.edu/~rjhala/liquid/

이 논문에서 초점을 두고 있는 것은 확장성(modularity)과 효과적인 증명 재구축(effective proof checker)입니다. 확장성을 가지기 위해 증명을 재구축하는 엔진을 각자 전공이 다른 여러 개의 작은 엔진들로 구성하였습니다. 효과적인 증명 재구축을 위해서는 coloring과 비슷한 방법을 사용하였습니다. SAT/SMT solver의 증명 과정에서는 많은 cluase들이 등장하는데, 앞으로 사용되지 않을 cluase를 메모리에서 제거함으로써 메모리를 효율적으로 사용하는 것이 핵심 아이디어입니다.

이론적으로 SAT/SMT solver의 파워를 이용할 수 있음을 보인 것이 아니라, 실제 그러기 위해서 어떠한 문제를 해결해야 하는가를 보여준 논문입니다. 이 정도라면 실제로 사용할 수 있지 않을까? 하고 생각하였습니다.

The Teaching Tool CalcCheck: A Proof-Checker for Gries and Schneider's "Logical Approach to Discrete Math".

Wolfram Kahl.

어느 날 점심식사를 하는데 한 교수님, Wolfram Kahl이 제 앞에 앉았습니다. 그 분은 증명체커를 이용하여 학생들에게 수학을 가르쳤다고 합니다. 교제는 "A Logical Approach to Discrete Math"였다고 합니다. 대학생들이 증명을 하면서 겪는 어려움 중 하나가 "내 증명이 정말 맞게 된 걸까?" 확인하는 작업입니다. 이를 도와주기 위해서 그 교수님은 CalcCheck라고 하는 증명체커를 만들었다고 합니다. 이 내용은 증명보조기를 이용하여 학생들을 교육한 사례를 공유하는 세션에서 발표되었습니다.

CalcCheck는 CPP에서 주로 언급되는 증명보조기들, Coq, Agda, Isabelle과는 성격이 다른 증명체커였습니다. 대부분의 증명보조기들이 귀납법이 기반하여 프로그램이나 어떤 수학적 성질을 엄밀하게 체크하는 데에 그 목적이 있었다면 CalcCheck는 학생들이 한 증명의 옳고 그름을 판별해 주는 데에 그 목적이 있었습니다. 이에 대한 주요 아이디어는 '모든 증명을 할 때에 꼭 귀납법을 사용해야 하는 것은 아니다, 대학교 저학년의 학생들이 이산수학을 배우는 데에 Coq이나 Isabelle과 같은 무거운 언어를 배우는 것은 부담일 수도 있다'는 것이었습니다. CalcCheck을 사용하기 위해서는 증명을 LaTeX로 작성하게 됩니다. 따라서 학생들은 꼭 의존타입 시스템(dependent type system)이나 Curry-Howard isomorphism과 같은 어려운 개념을 배우지 않고도 이산수학에서 필요한 올바른 증명을 배울 수 있습니다.

증명보조기라고 하면 Coq이나 Isabelle과 같이 원리를 이해하기 무거운 것들만 떠올렸었는데, 학생들의 시선으로, 그리고 목적에 따라 더 쉬운 해답을 얻을 수도 있다고 느꼈습니다. 또한, 순박하게 웃으시며 학생들을 가르친 경험을 이야기하시던 교수님의 모습이 기억에 오래 남아, '좋은 연구자'와는 또 다른 '좋은 교육자'에 대해서 생각할 수 있었습니다.

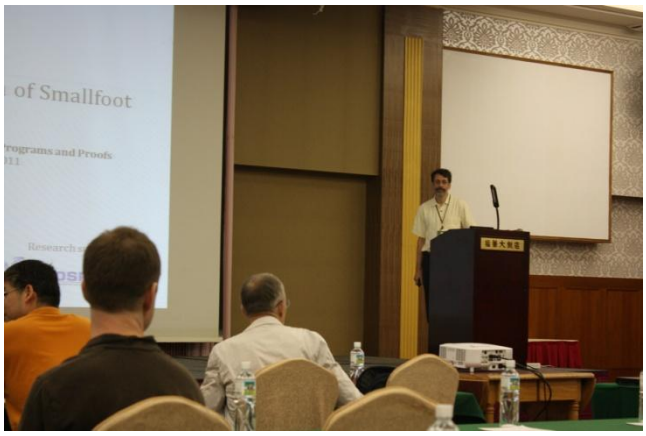
[INVITED TALK] VeriSmall: Verified Smallfoot Shape Analysis.

Andrew W. Appel.

ML 컴파일러로 유명한 Andrew Appel의 발표가 있었습니다. 대가의 발표라고 하여 모두가 기대하던 발표 중 하나였습니다. 이 발표에서 Andrew Appel은 검증된 모양분석기 VeriSmall을 소개하였습니다.

분석기의 소개에 앞서 검증된 소프트웨어 도구들의 사슬(verified software toolchain, 그림[3])에 대한 설명이 있었습니다. 우리가 소프트웨어 검증을 통해서 이야기하고자 하는 것은 다음과 같습니다.

"소스언어를 정적 분석기로 분석한 결과가 컴파일된 프로그램의 실행에서 그대로 나타난다."



위와 같은 주장을 하기 위해서는 사슬을 구성하는 소프트웨어 도구들이 모두 검증되고 연결되어야 합니다. 사슬은 크게 세 가지로 구성되며, 검증된 분석기 VeriSmall는 1에 해당하는 연구라고 할 수 있습니다.

1. 정적분석기 또는 프로그램 검증기
2. 컴파일러
3. 런타임시스템 또는 운영체제

ML코드의 추출

VeriSmall는 Coq으로 구현되었고, 그 안전성도 Coq으로 증명되어 있습니다. Coq의 추출(extraction) 기능을 통하여 Coq에서 정의된 프로그램을 ML코드로 뽑아낼 수 있습니다. 프로그램의 크기는 1,370LOC 이고 안전성 증명은 약 8,400LOC 입니다. 이는 검증된 C 컴파일러 CompCert의 1/6에 해당하는 규모입니다.

[3] Verified Software Toolchain, by Andrew W. Appel. *In ESOP 2011: 20th European Symposium on Programming*, LNCS 6602, pp. 1-17, March 2011.

종료 증명

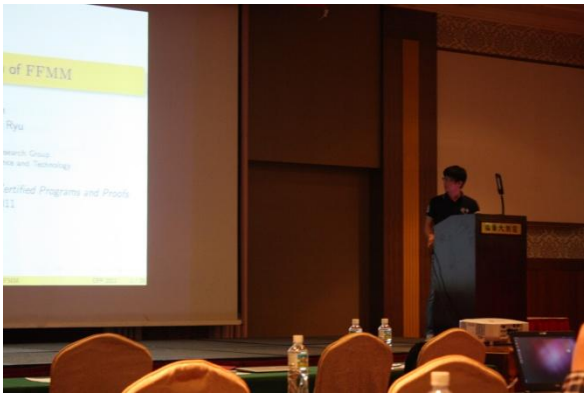
Coq으로 프로그램을 작성할 때 적지 않은 부담을 주는 것이 프로그램 종료에 대한 증명입니다. 이전에도 요약해석에 기반한 분석기가 Yves Bertot(Structural Abstract Interpretation: A Formal Study Using Coq)나 David Cachera 연구팀(A certified denotational abstract interpreter)에 의해 Coq으로 구현되었지만, 그때에도 종료를 증명하는 것에 부담이 있었습니다. 모양분석에서는 그를 어떻게 해결했을지 궁금했는데 아직 종료에 대한 증명은 완성되지 않았다고 합니다.

반복문 불변식(loop invariant)

VeriSmall은 반복문 불변식을 추론하지 못 합니다. 따라서 모든 반복문 불변식을 손으로 넣어 주어야 합니다. Algorithmic learning을 통해서 반복문 불변식을 찾는 방법이 ROSAEC 연구팀에 의해 연구되어 좋은 평가를 받고 있는데(TACAS'11, APLAS'10, VMCAI'10) 이 방법이 모양 분석기에서 사용될 수 있지 않을까? 하고 생각하였습니다.

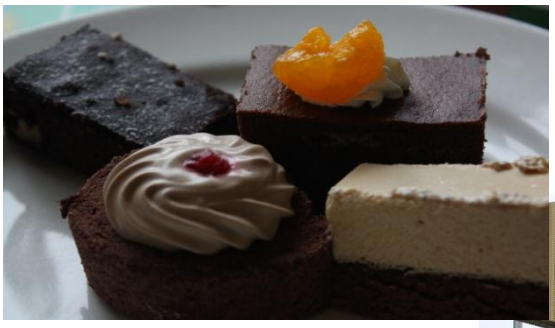
Coq Mechanization of Featherweight Fortress with Multiple Dispatch and Multiple Inheritance.

Jieung Kim and Sukyoung Ryu.

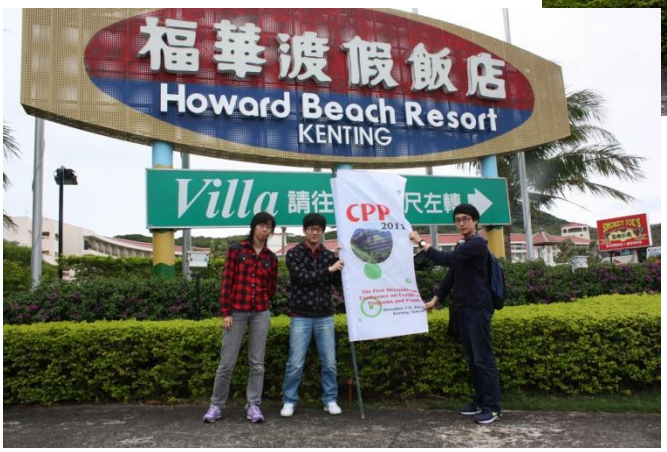


KAIST의 프로그래밍 언어 연구팀의 발표였습니다. 이미 ROSAEC 워크숍에서 여러 번 발표했던 내용이라 전체적인 내용은 이미 알고 있었습니다. 이 연구에서는 가벼운(featherweight) 포트리스 언어의 중요 계산법(core calculus)를 정의 및 기계화하고 타입 안전성을 Coq으로 증명하였습니다. 가볍다고는 했지만 함수나 메소드가 여러 개의 값을 가질 수 있는 multiple dispatch, 여러 부모 클래스를 가질 수 있는 multiple inheritance의 성질을 가지기 때문에, 이로 인해 생기는 문제를 정적으로 발견하여 해결해 줄 필요성이 있습니다. 타입 안전성의 증명내용을 구체적으로 확인하지는 않았지만 Coq으로 증명하기가 쉽지는 않았을 것이라 생각합니다.

이 발표는 김지응 학생의 첫 국제학회 발표였습니다. 저와 비슷한 시기에 대학원 공부를 시작했기에 매우 부러웠습니다. 발표를 들으면서 김지응 학생이 Coq을 배우겠다고 서울대학교의 이계식 박사님을 처음 찾아왔을 때가 생각났습니다. 도전적인 자세, 좋은 연구를 하기 위해서 꼭 필요한 것이 아닌가 싶습니다.




福華渡假飯店
Howard Beach Resort
KENTING
Villa
APLAS


福華渡假飯店
Howard Beach Resort
KENTING
Villa
CPP