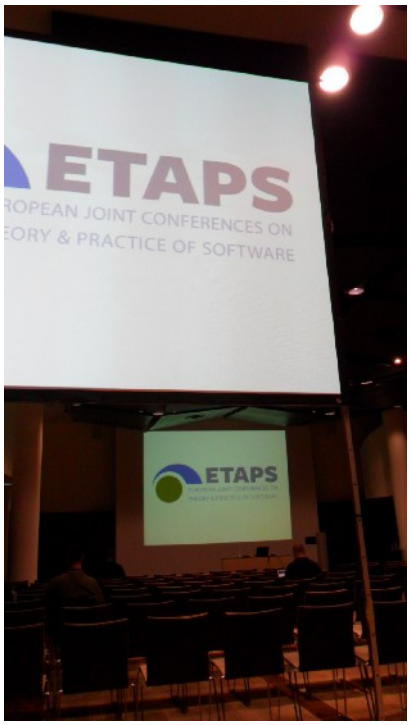
ETAPS 2012

European Joint Conferences on Theory & Practice of Software
26 March – 30 March 2012, Tallinn, Estonia

서울대학교 조성근



ETAPS는 유럽에서 열리는, 소프트웨어 전반에 대한 이론과 실제를 모두 아우르는, 합동(joint) 학회입니다. 다섯 개의 메인 학회(CC, ESOP, FASE, FOSSACS, TACAS)를 비롯하여 20여 개의 위성학회 및 워크숍이 함께 열렸습니다. 학회의 수로 보았을 때에 작지 않은 규모였지만, 사람 수는 지난 POPL에 비하면 많지 않았습니다.

이번 TACAS에 새로 시작된 POST(Principles of Security and Trust)라는 학회가 있습니다. 이름에서 알 수 있듯이 주로 보안에 관련된 주제를 다룹니다. 신생아 학회가 쟁쟁한 학회들과 함께 열려 인기가 없지는 않을까 생각했지만, 다른 발표장들에 비해서 큰 발표장을 할당 받았으며 사람도 많이 몰리는 것을 알 수 있었습니다. 보안에 대한 인기를 새삼 느낄 수 있었습니다.

에스토니아

올해의 ETAPS는 에스토니아에서 열렸습니다. 유럽에 처음가 봐서 그런지 에스토니아는 상당히 이국적이었습니다.

특히 커다란 돌맹이들로 만들어 놓은 알록달록한 찻길, 영화에서만 보던 그런 길이었습니다. 시내 중심가를 현대 건축이 비집고 들어오지 못 하도록 제한하여 아름다움을 간직하도록 해 놓았습니다. 우리나라로 따지면 안동과 같은 느낌이었습니다. 단, 안동에서는 오래된 건물들만 있고 현대인들의 실제 삶과 연결되어 있지 않은 느낌이 있지만 이곳의 마을엔 각종 잡화점, 음식점, 호텔 등이 들어서 있어 마치 과거 마을로의 시간여행을 한 것과 같은 느낌이 들었습니다.

에스토니아에는 전기로 가는 버스가 있습니다. 찻길 위에 전철처럼 전기선이 주렁주렁 메달린 모습이 인상적이었습니다.

에스토니아의 날씨는 상당히 추웠습니다. 3월 말인데도 불구하고 눈발이 날리는 험한 날씨였습니다.

음식은 너무나 괜찮았습니다. 유럽의 음식이 짜기만 하다는 소문을 들어 걱정을 하고 있었는데 그럴 필요가 없었습니다. 각종 소세지의 다양한 맛을 느낄 수 있었고, 소나 돼지와 같은 육류 뿐 아니라 생선도 상당히 많은 비율을 차지하고 있었습니다. 학회 참가 비용에 비해서 점심이 너무 잘 나왔고, 비싸지 않게 구한 호텔에서도 근사한 아침식사가 나와 지내는 내내 먹는 걱정은 없었습니다.

에스토니아의 택시는 값이 정해져 있지 않습니다. 택시를 탈 때마다 흥정을 하고 가격을 협상합니다. 이를 알고 있는지 택시 기사들은 항상 가격을 높게 부릅니다. 너무 비싸다며 돌아서면 바로 낮은 가격을 제시합니다. 택시의 가격이 정해져 있지 않다는 점이 신기했습니다. 가격 흥정이 자주 이루어지는 우리의 전통 시장과 같은 느낌을 받아 한편으로는 반가웠습니다.

발표

ETAPS에서 있었던 인상적인 발표를 정리하였습니다.

[Invited Talk] Foundations of C++. Bjarne Stroustrup.

C++ 언어를 만든 Bjarne Stroustrup의 초청강연이 있었습니다. C++은 단 2시간만에 설명하기는 어려운 복잡한 언어이지만, 발표는 주로 C++의 핵심에 대한 내용을 담고 있었습니다. C++이라 하면 현재 세계에서 가장 많이 사용되는 언어 중 하나이기에 어떤 내용을 이야기할지 발표에 관심을 기울였습니다.

하드웨어와 비슷한 언어

다른 더 높은 수준의 언어들과 비교했을 때, C++이나 C와 같은 언어의 가장 큰 특징은 프로그램이 하드웨어의 동작과 쉽게 대응된다는 점입니다. 물론 프로그래밍을 할 때 높은 수준에서 수학적인 개념만을 생각하는 것이 유리할 때도 있지만, 성능이 중요한 시스템 프로그램에서는 C나 C++과 같이 하드웨어의 동작을 직접 지정할 수 있는 언어가 더욱 유리하다는 것이 발표자의 주장이었습니다. 또한 이러한 저수준의 프로그래밍을 통해 더욱 효율적인 메모리 사용이 가능하다고 주장하였습니다. 이는 안전한 타입시스템을 만들 수 없는 이유와도 연관되어 있는데, 강제 타입 변환(cast)이나 미리 예측하기 어려운 배열의 접근에 의한 오류 등을 미리 막을 수 없는 이유가 되기도 합니다.

객체지향 그리고 generic 프로그래밍

C++에서 저수준의 프로그래밍이 가능한 반면 조금 더 높은 수준의 프로그래밍을 도와주는 것이 객채지향 프로그래밍과 generic 프로그래밍입니다. 발표자는 이 두 프로그래밍 방법이 서로 대치되는 것이 아니라 함께 어울려 사용될 수 있음을 강조하였습니다. 발표에서는 다형타입을 가지는 클래스를 예로 들었는데, C++의 표현력이 대단하다는 사실을 다시 한 번 느낄 수 있었습니다. 이러한 강력한 표현력이 정적 분석을 더욱 어렵게 하는 것이겠지요.

자원 관리

C로 프로그래밍을 할 때에 주의해야 하는 것 중 하나가 파일 또는 lock 등의 자원 관리입니다. 한정된 개수의 자원을 사용하거나 사용된 자원이 다시 회수되도록 할 때, 자원을 다루는(handle) 클래스의 도움을 받으면 더욱 안전한 프로그래밍이 가능하다고 합니다.

발표자는 메모리 관리(garbage collection)에 대해서 저와는 다른 시각을 가지고 있었습니다. 메모리는 단순한 자원으로 보기 어렵기 때문에 간단한 메모리 재사용 함수를 포함시키는 것이 메모리를 관리하는 하나의 방법일 수는 있어도 완전한 해결책이라고 보기는 어렵다는 주장이었습니다. 많은 함수형 언어에서 쓰이는 garbage collector가 미래에 사용될 수 없는 안전한 메모리만을 회수한다는 점을 생각하면 그러한 시각도 일리가 있다고 생각하였습니다. 오랜 시간 돌아야 하는 프로그램에서는 완전하지 않은 메모리 분석에 의해서 앞으로 사용되지 않을 메모리가 회수 또한 되지 않을 가능성이 있기 때문입니다.

[ESOP] The Call-by-need Lambda Calculus, Revisited, Stephen Chang and Matthias Felleisen,

이 연구에서는 새로운 call-by-need 람다 계산(calculus)를 제안하였습니다. 기존에 제안된 call-by-need 람다 계산들로는 Ariola 연구팀이 제안한 것과 Maraist 연구팀이 제안한 것이 있습니다. 이 연구에서는 기존의 두 call-by-need 람다 계산에서 발견되는 문제점들을 보완하였습니다.

기존 람다 계산의 가장 큰 문제점은 호출된 함수가 사라지지 않는다는 점입니다. 예를 들어, call-by-value나 call-by-name에서는 함수를 인자에 적용하면 다음과 같이 람다가 하나 사라지게 됩니다.

$$(\lambda x.e)\ v \rightarrow e[x\backslash v]$$

그러나 call-by-need에서는 람다가 사라지지 않습니다. e에 포함된 x 중 어느 것이 실제로 필요한지 모르기 때문입니다. 따라서, 다음과 같이 실행 문맥(evaluation context)을 이용하여 실제 필요한 x의 값만을 하나씩 다시 쓰게(translation) 됩니다.

$$(\lambda x.C[x])\ v \rightarrow (\lambda x.C[v])\ v$$

Maraist 연구팀의 람다 계산 규칙에는 다음과 같이 람다를 제거하는 규칙이 있어 문제를 해

결하고자 했지만, 항상 적용 가능한 것이 아닙니다.

$$(\lambda x.e)\ e' \rightarrow e \text{ when } x \notin fv(e)$$

Chang 연구팀은 문맥에 명시적으로 필요한 계산의 개념을 추가하여 문맥에 의한 문법으로 이 문제를 해결하였다고 합니다. 이에 대한 내용이 빠르게 지나가 모두 이해하지는 못하였지만 흥미로운 내용이었습니다. 이미 call-by-need가 나온 지 거의 20년이 가까이 되어가는데 이에 대한 연구가 꾸준히 진행되고 있다는 사실이 놀랍습니다.

[ESOP] Generate, Test, and Aggregate — A Calculation-based Framework for Systematic Parallel Programming with MapReduce. Kento Emoto, Sebastian Fischer and Zhenjiang Hu.

훌륭한 발표였습니다. 많은 발표가 너무 어려운 수식을 설명하거나, 너무 많은 내용을 담으려는 이유로 이해를 어렵게 했던 반면, Emoto의 발표는 연구의 핵심이 되는 발표를 천천히, 그리고 그림과 함께 직관적으로 설명하였습니다. 발표를 듣는 사람들이 발표 내용을 이해했는가 그렇지 못 했는가는 발표하는 사람의 수를 보면 알 수 있는데, 발표가 끝나고 정말 많은 사람의 질문이 있었습니다. 발표 능력이 꼭 유창한 영어 실력과 비례하지는 않는다는 것을 느낄 수 있었습니다.

이 연구에서 풀고자 하는 것은 '어떤 문제들을, 어떻게 효율적인 MapReduce 형태로 만들 수 있을까?'입니다. MapReduce는 병렬 프로그래밍을 위한 프레임웍으로서, 큰 문제를 작은 문제로 쪼개는 map step과 여러 결과들로부터 최종 결과를 만드는 reduce step으로 나눌 수 있습니다. 지금까지 MapReduce 형태의 프로그램을 만드는 방법에 대한 연구들은 거의 이루어지지 않았고, 몇몇 연구들이 특별한 경우에 대한 제한적인 방법만을 제시했다고 합니다. 따라서 이 연구에서는 '어떤 문제들이 MapReduce의 형태로 바뀔 수 있을까?', '어떻게 하면 효율적인 프로그램을 만들어 낼 수 있을까?'에 초점을 두고 있습니다.

Emoto 연구팀은 GTA형태(generate, test, aggregate)를 갖는 모든 프로그램에 대해서 자동으로 효율적인 MapReduce 프로그램을 생성하는 방법을 제안하였습니다. 세 단계를 설명하면 다음과 같습니다.

- generate: 해가 될 수 있는 모든 후보 생성
- test: 제약조건에 따라 해가 될 수 없는 후보 삭제
- aggregate: 남은 가능한 해 중 가장 좋은 해를 선택

핵심 아이디어는 GTA형태의 문제를 divide and conquer 문제로 변환하는 것에 있습니다. 그리고 divide and conquer 문제는 쉽게 MapReduce 형태로 변환할 수 있다고 합니다. 발표자는 예를 들어 이런 변환들을 설명하였습니다. 인상적인 발표였고 나중에 차근차근 논문을 읽어 보고 싶다고 생각하였습니다.

[ESOP] GMeta: A Generic Formal Metatheory Framework for First-Order Representations. Gyesik Lee, Bruno C.d.S. Oliveira, Sungkeun Cho, and Kwangkeun Yi.

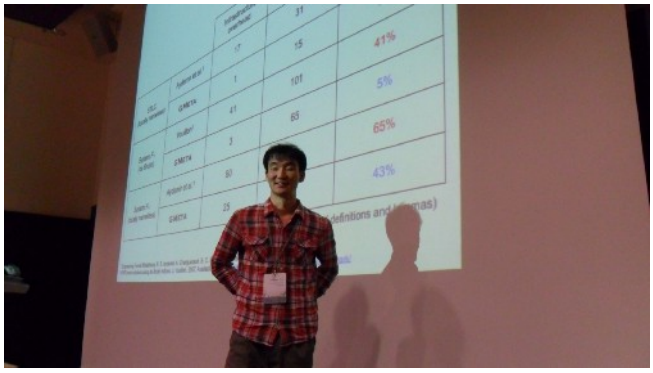
기계적인 증명을 작성할 때 변수묶임(variable binding)에 관련된 성가신 정의 및 정리들의 부담을 줄여 주는 연구입니다. 그런 성가신 정의의 예로 free variable 또는 bound variable에 대한 substitution을 들 수 있습니다. 이러한 함수는 대부분의 언어에서 거의 기계적으로 정의 가능하지만 언어가 수정될 때마다 새롭게 정의해 주어야 합니다. 이러한 것들을 중요한 정의를 위한 기반이 된다는 의미에서, infrastructure라 부르는데, 프로그래밍 언어를 대상으로 다루는 많은 증명에서 infrastructure의 비율이 상당히 높습니다. 우리의 실험에서는 모두 40% 이상이었습니다.

GMeta는 전체 증명 중 infrastructure의 비율을 평균적으로 35% 줄여 주는 infrastructure 생성 라이브러리입니다. 사용자로부터 사용자가 정의하고자 하는 언어를 입력으로 받으면 그 언어로 된 프로그램에 적용 가능한 infrastructure를 자동으로 생성해 줍니다. 사용자가 GMeta를 사용하기 위한 비용은 거의 없으며, 심지어 GMeta의 동작 원리에 대해 알고 있다면, 사용자가 정의하려는 언어에 특화된 더 많은 infrastructure를 얻어낼 수도 있습니다.

발표는 이계식 교수님과 나누어서 하였습니다. 국제학회에서의 첫 발표였기에 조금은 긴장되었습니다. 제게 주어진 시간은 10분이었고 저는 GMeta를 이용한 실험 내용에 대해 발표하였습니다. 발표를 위한 대본을 미리 준비했기에 발표 도중 버벅거리는 일은 없었지만, 제한 시간을 꼭 지켜달라는 세션체어의 당부가 있었기에 저도 모르게 발표를 급하게 진행한 것이 마음에 걸립니다. 목소리는 별로 떨리지 않았지만, 심하게 떨리는 레이저 포인터를 보며 스스로 긴장했다는 사실을 실감할 수 있었습니다.

발표를 준비하고 발표를 진행하면서 느낀 점은 다음과 같습니다.

1. 청중들을 보면서 발표하자.



짧은 발표를 하면서, 그리고 다른 사람들의 발표를 들으며 느낀 점은 발표가 단순히 나의 연구 내용을 뽐내는 시간이 아닌, 그것을 청중들에게 이야기하는 대화라는 점입니다. 종종 슬라이드만 바라보며 발표하는 발표자들이 있었는데 그들의 공통점은 슬라이드의 내용이 짧은 시간에 이해하기에 상당히 어렵다는 점입니다. 슬라이드에 적힌 내용이 너무 전문적이고 어려워서 당연히 청중들을 바라보며 발표할 수 없게 되는 것입니다. 청중들을 바라보며 진행하는 발표를 준비하는 것은 이해하기 쉬운 발표와 연결될 가능성이 높다고 느꼈습니다.

2. 제한된 시간을 지키자.

제한된 발표 시간에 맞추어 발표를 연습하는 것도 위의 내용과 같은 맥락으로 이해할 수 있습니다. 몇몇 발표자들은 발표 시간이 부족하여 몇 장의 슬라이드를 그냥 넘기기도 하였는데 그러한 발표 또한 발표 내용이 전문적이고 어려웠습니다. 발표를 시간에 맞출 수 있도록, 간결한 어휘로 쉽게 설명하는 발표 준비를 한다면 준비된 내용을 모두 전달하는 데에 도움이 될 것이라 생각합니다.

3. 못난 발표에도 집중하자.

어떤 발표를 들으며 이해가 잘 안된다거나 못난 발표라 느끼는 것에는 그럴만한 이유가 있습니다. 너무 말이 빠르다든지 청중이 알아듣기 어려운 전문 용어를 설명 없이 사용한다든지... 훌륭한 발표를 보고 좋은 점을 찾는 것도 좋지만, 못난 발표를 들으며 주의해야 될 점을 골라내는 것이 더 쉬울 수 있습니다.

4. 리허설은 빠른 시기에 하자.

저는 출국 2주 전, 연구실 내부 세미나 시간에 리허설을 하였습니다. 덕분에 리허설 때에 받았던 지적과 수정 사항들에 대해 고민할 수 있는 충분한 시간을 가질 수 있었습니다.

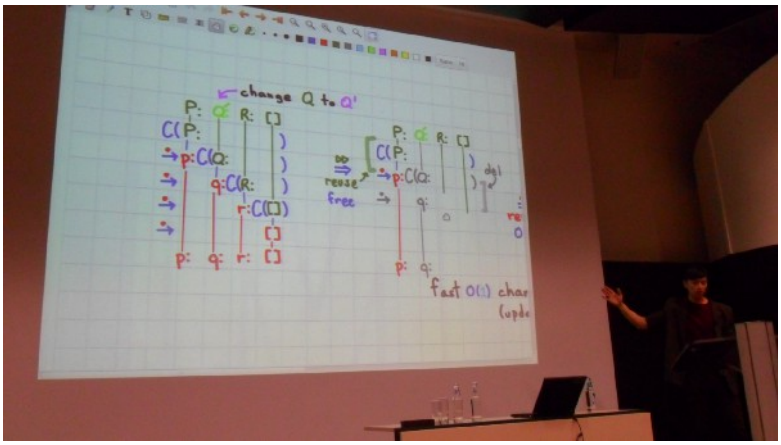
[ESOP] Non-monotonic Self-Adjusting Computation, Ruy Ley-Wild, Umut A. Acar, and Guy Blelloch.

이 연구는 어떤 프로그램의 실행을 재사용하는 방법에 대한 것입니다. 즉, 프로그램이 어떤 입력값에 의해서 실행된 후, 다른 입력값이 들어 왔을 때 처음부터 다시 실행하는 것이 아니라 이전 출력값을 업데이트하는 방식으로 이전에 수행했던 실행을 재사용합니다. 이것이 단조(monotone) 프로그램에 대해서는 어렵지 않은 해결책이 나와 있지만, 그렇지 못한 프로그램에 대해서는 이전 프로그램의 실행을 크게 수정해야 하는 경우가 있습니다.

출력값 업데이트의 핵심은 프로그램 실행 조각(trace slices)를 이용하는 것입니다. 실행 조각에는 데이터 의존관계(data dependency)나 실행 의존관계(control dependency)가 포함되어 있고, 이것이 실행 조각의 순서를 재구성하는 데에 사용됩니다. 그리고 이 실행 조각 순서의 재구성은 단조가 아닌 프로그램의 실행 재사용에 사용되게 됩니다.

어떤 프로그램을 실행할 때 유사한 입력으로 비슷한 실행을 다시 수행하는 경우는 많습니다. 컴파일러의 경우 어떤 소스코드를 컴파일하고, 수정 후, 다시 컴파일하는 경우는 부지기수입니다. 이는 프로그램 분석의 경우도 마찬가지입니다. 분석을 수행하고, 오류를 수정한 후, 프로그램 분석을 다시 수행할 수 있습니다. 이런 경우 이전의 분석 결과를 효과적으로 재사용할 수 있다면 좋을 것 같다고 생각하였습니다.

이 연구에서 한 가지 아쉬운 점은 실험 결과가 없었다는 점입니다. Ruy Ley-Wild 연구팀은 프로그램 재사용에 적합한 저수준(low-level)의 언어를 디자인하고, 고수준(high-level) 언어에서 저수준 언어로의 변환을 디자인하였으며, 이것이 옳음을 보였습니다. 하지만 실제로 이러한 언어를 사용하여 얼마나 많은 실행 비용을 줄일 수 있었는지에 대한 실험 내용은 없었습니다. 이전에 실행된 프로그램 실행 조각의 길이가 n이라고 했을 때, 이를 재사용하기 위한 비용이 O(log n)이라고 하였지만, 이것이 실행 재사용에 의한 비용 절감에 얼마나 영향을 끼칠지 알 수 없었습니다.



발표 슬라이드는 특이하게도 타블렛을 사용하여 손으로 작성되었습니다. 발표자는 타블렛이 아니고서는 표현하기 어려운 다양한 그림을 손으로 그려 발표하였습니다. 하지만 두 가지 문제점이 있다고 생각했는데, 손으로 쓴 수식은 눈에 잘 들어오지 않는다는 점과 타블렛으로 그려야만 할 정도로 복잡한 그림은 애초에 짧은 발표에서 설명하기에 적합하지 않을 수 있다는 점입니다.

[TACAS] Competition on Software Verification.

TACAS(Tools and Algorithms for the Construction and Analysis of Systems)라는 이름에 걸맞게, 논문의 발표가 아닌 소프트웨어 분석기 대결 세션이 있었습니다. 여러 종류의 벤치마크 프로그램을 제출된 분석기로 분석하고 그 결과에 점수를 매겨 승자를 가리는 대결이었습니다. 분석 대상은 정수값 관련 프로그램, 드라이버(32/64bit), 힙 메모리 관련 프로그램, 시스템 프로그램, 병렬 프로그램으로 나뉘어졌고, 총 10개의 분석기가 제출되어 비교되었습니다. 분석기는 모델검사, SMT solver, 분리논리 등 다양한 이론에 기반하고 있었습니다. 각 분석기에 대한 소개와 시상이 함께 이루어졌습니다. 제출된 분석기 중 인상깊은 분석기를 소개하겠습니다.

BLAST 2.7

이 분석기는 C언어의 모델검사를 수행하는 BLAST 2.5(오픈소스)을 개량한 분석기입니다. 개량은 주로 속도 향상에 초점이 있었고 기존의 BLAST보다 8~30배 빨라졌다고 합니다. 분석기의 목표가 리눅스 드라이버 검증에 있었기 때문인지 드라이버 관련 대결에서 높은 점수를 받았습니다.

http://forge.ispras.ru/news/68

Predator

이 분석기는 분리논리에 기반한 분석기입니다. 분리논리에서의 논리식을 그래프로 표현하고 이를 이용하여 분석합니다. 그래서인지 힙 메모리 분석에서 가장 높은 점수를 받았습니다. 이 분석기는 GCC의 플러그인으로 구현되어 있습니다. 이는 GCC로 파싱 가능한 모든 프로그램에 대해 분석이 가능하여 매우 실용적이라고 합니다.

http://www.fit.vutbr.cz/research/groups/verifit/tools/predator/

ESBMC 1.17

이 분석기는 모델검사에 기반하고 있고 분석을 위하여 SMT solver를 이용한다는 특징을 가지고 있습니다. 또한 멀티쓰레드에 의한 상호 간섭을 효율적으로 풀어내고 이를 분석하는 기술을 이용하므로, 병렬 프로그램의 분석에 강점을 가지고 있다고 합니다. 8개의 병렬 프로그램 중 무려 6개의 프로그램 분석에 성공하여 이 부문에서 가장 높은 점수를 받았습니다. 참고로 병렬 프로그램 분석에서는 다른 한 분석기가 하나의 케이스에 대해 성공하였고, 다른 모든 분석기는 병렬 프로그램을 전혀 분석하지 못 하였습니다.

http://www.esbmc.org/

QARMC-HSF

이 분석기는 CEGAR(CounterExample-Guided Abstraction Refinement)에 기반하고 있습니다. 검증될 프로그램과 검증에 사용될 증명 규칙을 Horn clause 형식으로 기술하고, 이를 입력으로 받아 분석을 수행한다는 특징을 가지고 있습니다. 분석기를 Prolog 언어로 구현했다는 점이 특이한데, 분석 과정에서 선형 연산(linear arithmetic)이 만족되는가를 검사하기 때문이라고 합니다.

http://www7.in.tum.de/tools/hsf/

각 분석기의 발표가 짧아(7분) 많은 부분을 이해하지는 못 했지만, 다양한 분석 방법이 존재하고 세계 각국에서 다양한 시도가 이루어짐을 확인할 수 있는 유익한 시간이었습니다.

[TACAS] Zeno: An Automated Prover for Properties of Recursive Data Structures, William Sonnex, Sophia Drossopoulou, and Susan Eisenbach.

Zeno는 자동으로 증명을 생성하는 도구입니다. 이 도구의 입력은 Haskell로 짜여진 함수와 그에 대한 성질들이고, 출력은 그 함수의 주어진 성질들에 대한 Isabelle 증명들입니다. Zeno는 뛰어난 휴리스틱으로 증명을 탐색하는데 자동 증명 도구 ACL2와 비교해도 더 우수한 성능을 보인다고 합니다.

Zeno에서 사용한 휴리스틱은 다음과 같습니다.

- 반례를 이용하여 증명 공간 탐색에 필요한 비용을 줄입니다. 즉, 어떤 증명 규칙을 이용하여 여러 개의 증명 목표(goal)을 만들었을 때, 이 중 하나라도 반례가 발견된다면 적용된 증명 규칙이 적절하지 않음을 의미합니다. 이는 이미 많이 알려진 방법으로 SmallCheck이나 ACL2s에서도 사용하고 있다고 합니다.
- 이전에 시도하여 실패했던 유사한 증명 과정을 피하는 방법을 사용하여 빠른 공간 탐색을 수행합니다. 즉, 기존에 시도했던 모든 증명 과정을 기억함으로써, 유사한 증명 과정이 시작될 때 이를 피할 수 있게 됩니다.

Zeno는 지금까지 증명된 보조 정리를 사용할 수 없어 커다란 증명에 대해서는 ACL2s가 더욱 효과적이지만, 간단한 증명에 대해선 높은 성능을 보이므로 Isabelle 증명 보조기에 통합시키는 것이 다음 목표라고 합니다. 자신의 연구에 대한 솔직한 평가가 인상적이었습니다.

아래의 페이지에서 Zeno를 사용해 볼 수 있습니다.

http://www.doc.ic.ac.uk/~ws506/tryzeno/

감사합니다.

연구를 열심히 해야겠습니다. 연구 내용이 부족한 학회 참가자의 고민은 학회장에 가면 모르는 사람들과 할 얘기가 없다는 것입니다. 학회 참가는 다른 분야의 사람에게 나의 연구 내용에 대해 이야기하고 피드백을 받을 수 있는 좋은 기회입니다. 이러한 대화는 연구 내용과 배경지식을 간결하게 설명하는 것을 수반하기 때문에 논문 작성 또는 발표 준비와 크게 다르지 않습니다. 하지만 연구한 것이 없으면 이러한 기회가 무용지물이 되기 일쑤입니다. 이번 기회에 다시 한 번 연구를 열심히 해야겠다고 생각했습니다.

먼저 학회 참가를 지원해 주신 이광근 교수님과 ROSAEC센터에 감사드립니다. 또한, 전체 발표 시간 중 일부를 허락하여 발표의 기회를 주신 이계식 교수님께도 감사드리며, 발표 준비에 조언을 아끼지 않은 연구실 선후배분들께도 감사드립니다. 앞으로도 서로 좋은 피드백을 주고받으며 훌륭한 연구를 할 수 있기를 바랍니다. 감사합니다.