APLAS + CPP 2012

12.4 – 12.9 KENTING, TAIWAN

이우석 (서울대학교 프로그래밍 연구실)

이번 아플라스는 이번에 처음 열리는 국제학회 CPP(Certified Programs and Proofs)와 함께 열리게 되었다. 두 학회 합해서 총 6 개의 초청강연과 47 여개의 발표, 그리고 2 개의 튜토리얼이 있었다.

학회가 열린 위치는 대만에 Kenting 이라는 곳인데, 대만 내에서는 유명한 휴양지이다. 우리나라로 치면 제주도쯤 된다고 한다. 이곳은 1984 년부터 국가보호구로 지정되어서 그런지 자연환경이 정말 깨끗하긴 했지만 볼 거리가 많지는 않았다.

학회는 전반적으로 유익하고 즐거웠다. 초청강연은 Peter O'hearn, Andrew Appel 등 유명한 대가들이 준비하여 사람들의 관심을 모았고, 흥미로운 연구발표들도 많이 있었다.

흥미로웠던 발표/초청강연/튜토리얼 몇 가지를 소개할까 한다.

APLAS

TUTORIAL 1: DEPENDENTLY TYPED PROGRAMMING IN ADGA

Dependent type 이란 어떤 값으로 표현되는 타입을 말한다. 예를 들어 정수의 여러쌍으로 표현되는 Vector 라는 타입이 있을 때 Vector n 을 n 개의 정수 쌍 (int x int x ... x int)으로 표현할 수 있다. 이 때 타입이 자연수 값 n 에 dependent 하다고 한다. Dependent type system 은 Coq, Agda, Haskell 등의 언어들에 구현되어있다.

이러한 dependent type 은 기존에 타입으로 표현할 수 있는 범위를 넓혀서 기존 타입 시스템으로 증명되기 어려웠던 성질들에 대한 증명이 가능케 하는데 주된 이점이 있다. 아주 단순한 예로 벡터의 내적을 구하는 함수 inner 를 정의한다고 하자. 예전에는 inner : Vector -> Vector -> Nat 로 정의하던 것을 inner : (n : Nat) -> Vector Nat n -> Vector Nat n -> Nat 으로 표현할 수 있다(여기서 Vector Nat n 은 길이 n 짜리의 벡터 타입을 의미한다). 즉 인자로 받는 두 벡터가 길이가 같아야 한다는 제약을 추가로 줄 수 있다. 여기서 괄호 친 첫번째 인자 n 은 생략 가능하다.

튜토리얼에서는 Agda 라는 함수형 언어에서 아주 간단한 가상 스택머신을 정의하고, 아주 간단한 언어를 정의한 후, 그 언어로 쓰인 프로그램을 가상 스택머신 언어로 바꾸어주는 컴파일러의 안전성을 dependent type 시스템으로 증명하는 과정을 보였다. 타입에 맞게 컴파일러 코드를 작성하면 그 자체로 컴파일러의 올바름이 보장되었다.

A NON-ITERATIVE DATA-FLOW ALGORITHM FOR COMPUTING LIVENESS SETS IN STRICT SSA PROGRAMS

데이터 흐름 분석에서 liveness 분석이란 각 프로그램 지점마다 살아있는 변수(live variables)를 계산하는 분석이다. 한 지점에서 살아있는 변수란 그 전에 정의되었고, 그 지점 이후 지점에서 사용되는 변수이다. 이 정보는 컴파일러 뒷단에서 레지스터 할당, 쓸데없는 코드 없애기, 소프트웨어 파이프라이닝(반복문 최적화 기법 중 하나) 등의 여러가지 최적화를 하는데 사용된다.

보통 데이터 흐름 분석에서 분석 결과는 고정점 계산(fixpoint iteration)으로 구한다. 살아있는 변수를 계산하는 일반적인 방법은 어떤 지점에서 사용된 변수에 대해서, 그 지점부터 그 변수가 정의된 지점까지 위로 거슬러 올라가면서 거쳐온 지점들에 이르는 범위에서 해당 변수가 살아있는 변수라고 결론짓는 것이다. 기존 방법은 이것을 고정점에 도달할 때까지 반복하는데, 루프가 여러 단계 중첩되어 있거나, 크면 분석이 오래 걸릴 수 있다.

이 논문에서는 고정점 계산을 하지 않고 살아있는 변수들을 구하는 방법을 제시한다. 실험 결과 그들의 방법이 고정점을 계산하는 알고리즘에 비해 평균 2 배 정도 빠르다고 한다. 방법은 다음과 같다. 우선 프로그램을 SSA(static single assignment) form 으로 바꾸고(모든 변수는 단 한번 정의된다. x=... x=... 와 같이 여러 번 정의될 경우 서로 다른 첨자를 부여하여 다른 변수로 만듦) 다음 두 단계를 거친다.

1. 루프 간선(실행 흐름 그래프에서 루프 안의 노드에서 루프 헤드로 이어지는 간선)를 없애고 모든 블록에 대해서 고정점 계산 방법에서 하는 것처럼 변수 정의 지점까지 위로 올라가면서 살아있는 변수들을 구한다.
2. 루프 헤드(루프 입구)에 모여있는 살아있는 변수들에 대한 정보를 루프 안의 노드로 전파시킨다. 이 때 SSA 의 Φ-function 에 의해 정의된 변수들은 제외하고 전파시킨다.

이렇게 하는 이유는 다음과 같다 : 우선 1 번에 의해서 루프를 제외한 부분에 대해서는 분석이 완료된다. 그리고 루프 내부에서는, 루프 안에서 정의된 변수들에 대해서는 liveness 정보가 루프 안 노드에 대해서 제대로 퍼져있지 않은 상태이다. 이를 위해 2 번과 같은 일을 하는데, 단순히 루프 헤드에 모인 정보를 다 아래로 전파시키지 않고 Φ-function 에 의해 정의된 변수들은 빼는 이유는, Φ-function 으로 정의된 변수들은 1 단계에서 잘 처리되기 때문이기도 하고, 이것들을 루프 안에 다 전파시키면 쓸데 없는 전파로 결과가 더 부정확해지기 때문이다.

논문에서는 위의 방법이 올바르다는 사실 또한 증명하였다.

ACCESS-BASED LOCALIZATION WITH BYPASSING

같은 연구실의 오학주 연구원이 발표하였다(이하 학주 형). 이 논문의 내용은 연구실 정적 분석기 Airac 을 지역화(localization)를 통해 최적화 한 내용과 최근의 성긴 분석(sparse analysis)를 통해 최적화한 두 성과의 중간단계 성과이다.

결론적으로 발표는 학회에서 가장 성공적인 발표 중 하나였고, 어찌 보면 정말 기초적일 수 있는 다음 교훈들을 얻었다.

1. 발표자 스스로가 재미있어야 청중들이 흥미를 갖고 듣는다.
2. 이 논문이 무엇을 어떻게 한 건지 도입부에 명료하게 설명해야 한다.

처음에 학주 형은 이 논문이 이미 1 년전의 연구 내용에 대한 것이고 지금은 더 개선된 결과를 갖고 있어서 발표에 별 흥미가 없었다고 한다. 그런데 연구실 내부 세미나에서 발표 리허설 때 교수님께서 스스로가 재미있어야 청중들도 흥미를 갖고 듣는다고 하시면서, 우리가 분석기를 빠르게 하기 위해 그 동안 어떻게 해왔고 최근엔 어떠한 상태이며 이것은 그 중간 단계에 대한 내용임을 발표 도입부에서 얘기하라고 하셨다. 학주 형은 이와 같은 흐름을 갖게 슬라이드를 고쳤고 발표를 성공적으로 마쳤다. 만약 전후 맥락 없이 연구 내용만 발표되었다면 청중의 흥미를 못 끌었을 뿐 아니라 발표자 또한 재미 없었을 것이다. 연구의 전후 맥락을 시작 부분에서 얘기함으로써 청중에겐 우리 연구그룹이 어떤 일을 하는 집단인지 명확히 얘기하면서 주의를 환기시킬 수 있었고 동시에 이 연구가 무엇을 한 것인지 초반에 명확히 얘기할 수 있었다.

분리 논리(separation logic)로 유명한 Peter O'hearn 이 발표에 대해 좋은 평가를 했는데, 전하고자 하는 바가 명확하고 이해하기 쉽기 때문일 뿐만 아니라 이 논문이 Peter O'hearn 의 꿈에 대한 가능성을 보여준 것이기 때문이다. Peter O'hearn 은 분리 논리에 지역화에 대한 가이드라인을 포함시켜 정말 실용적인 논리 시스템을 만들기를 원한다고 한다.

COST ANALYSIS OF CONCURRENT OO PROGRAMS

Cost analysis 는 주어진 입력 패러미터에 대해서 프로그램이 얼마만큼의 많은 자원(시간, 메모리 등)을 사용하는지 알아내는 분석이다. 이 논문은 그 동안의 cost analysis 가 병렬 프로그램에 대해서는 연구가 진행되어오지 않았다고 문제제기를 하면서 병렬 프로그램에 대한 cost analysis 틀을 제시한다. 논문에서 제시하는 분석의 장점은 각 instruction 에 대응되는 cost model 을 무엇으로 하느냐에 따라 종료 분석, 메모리 소비량 분석 등 여러 분석을 할 수 있고, 각 함수별로 어느 정도의 자원을 소모하는지 정량화하여 서로 비교해 볼 수도 있다는 점이다. 저자들은 본 논문에 제시된 이론적 틀에 기반한 구현체도 제시하고 있는데 온라인으로 접근해서 직접 해볼 수 있다.

그러나 아직은 작고 간단한 프로그램에 대해서만 자동 분석이 가능한 수준이다. 사이트에 제시된 크지 않은 예제 프로그램들에 대해서도 invariant 가 주석으로 분석기에 제공되고 있고, 이들 중 일부를 지우면 분석이 되지 않는다.

CPP

SIMPLE, FUNCTIONAL, SOUND AND COMPLETE PARSING FOR ALL CONTEXT-FREE GRAMMARS

이 논문은 CFG parser generator 에 대한 내용이다. 함수형 언어를 이용해서 파서를 만들 때 매우 인기있는 방법으로 조합적 파싱(combinatory parsing)이라는 것이 있다. 이것은 간단한 문법에 대한 파서 여러 개를 조합하여 복잡한 문법에 대한 파서를 만드는 방법이다. 예를 들어, E → E E E | "1" | ε 와 같은 문법이 주어질 때 다음과 같이 parser 를 구현할 수 있다.

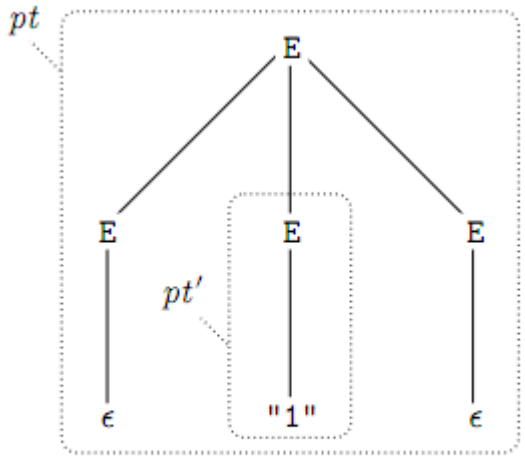
```
let rec E = fun i → ((E **> E **> E) ||| (a "1") ||| (a "")) i
```

여기서 **> 와 ||| 는 파서를 받아서 파서를 돌려주는 고차 함수이고 a 는 말단기호(terminal symbol)를 소비하여 내부적으로 파서의 상태전이를 일으키는 함수이다. 이렇게 쓰면 문법을 정의하는 것과 거의 비슷하게 코드를 짤 수 있기 때문에 쉽다.

그런데 이렇게 파서를 만들 때 주의해야 할 점 하나가, 끝나지 않는 파서를 만들 수도 있다는 점이다. 위의 문법은 left-recursive 문법 (비말단기호(nonterminal symbol)가 왼쪽 끝에 나오는 문법)이므로 끝나지 않을 수 있다 (왜냐하면 E 를 E E E 인 경우로 처리하고, 다시 왼쪽 끝 E 에 대해서 E E E 로 처리하려고 하는 과정이 반복되기 때문).

이 논문의 주된 내용은 늘 끝나는 파서를 생성하는 파서 생성기에 대한 것이다. 이들은 좋은 parse tree, 나쁜 parse tree(끝나지 않는 parse tree)란 무엇인지 formal 하게 정의하고, 이 논문에서 제시하는 parser generator 에 의해 만들어진 parser 가 늘 좋은 parse tree 만 생성한다는 것을 HOL4 라는 증명 도구를 이용하여 증명하였다.

이들 parser generator 의 핵심 아이디어는 들어온 인풋에 대해서 생성될 수 있는 많은 parse tree 중에서 나쁜 parse tree 를 제거하는 것이다. 나쁜 parse tree 란 별 게 아니고, 쓸데없는 비말단 기호를 확장하는 짓이 일어나는 트리이다. 예컨데 아래 파스 트리는 위에서 예제로 든 문법으로 문자열 "1"을 파싱한 결과인데 이것은 나쁜 파스트리이다.



substring_of pt = substring_of pt'

왜냐하면 한 번 이상 아무 의미없이 비말단 기호를 확장한 파스트리이기 때문이다. (의미없다는 것은 같은 비말단 기호를 생성해내고, 또 동시에 어떤 말단 기호도 소비하지 않는 것을 의미한다. 위의 예에서 E → E E E 가 그렇다) left-recursive 문법에서 무한 루프는 결국 이러한 일이 반복되기 때문에 일어난다. 이러한 상태는 어떤 파스트리에 대해서 다음의 조건을 만족하는 서브 파스트리가 존재하는지 검사함으로써 발견할 수 있다.

1. 파스트리와 같은 비말단 기호를 루트에 둠
2. 파스트리에서 처리되는 문자열과 똑같은 문자열을 처리함.

위의 그림 예제에서 주어진 파스트리 pt 에 대해 이 조건을 만족시키는 서브트리가 pt'이다.

이 논문의 주된 기여는 문제를 formal 하게 정의하고 기존 방식의 옳음을 증명한 데 있다. 사실 이러한 해결법은 기존에 있었으나, 알고리즘의 옳음(soundness)이 증명되지 않았다. 또한 이 논문은 알고리즘이 완전하다는 것(completeness)또한 증명하였다. 완전하다는 것은 주어진 문법으로 파싱 될 수 있는 모든 문자열이 파서에 의해 파싱된다는 것이다.

또한 이 논문은 자신들의 파서 생성기와 Haskell 파서 생성기 Happy 와 성능비교를 통해 자신들의 것이 더 우수함을 보였다. 나는 예전에 elkhound 라는 CFG parser generator 를 사용해 본적이 있는데, 이 파서 생성기는 C++로 짜여지고 온갖 최적화 기법을 다 동원한 것으로 알려져있다. 아마 조합적 파싱기술에 기반한 파서 생성기가 아니라서 비교대상에서 제외했을 수는 있겠지만 elkhound 를 포함하여 더 많은 CFG 대상 파서 생성기와 성능을 비교했더라면 논문이 더 흥미로웠을 거란 생각이 들었다.

발표는, 전반적으로 성과에 대한 자부심이 느껴지는 좋은 발표였지만 기술적인 세부사항에 대해서는 전달이 좋지 못했다. 초반에 문제가 무엇인지는 아주 잘 설명한 것 같다. 그러나 (슬라이드를 Prezi(http://prezi.com/index/)로 만든 것 같다) 초반에 관심을 끌었던 줌과 이동 등의 화려한 애니메이션이 나중엔 산만하게 느껴졌고, 뒷 부분에는 논문을 거의 복사해놓은 슬라이드들이 있어 기술적 내용을 이해하기는 어려웠다.

ENGINEERING THEORIES WITH Z3

MSR 의 Nikolaj Bjoner 의 초청발표이다. 발표의 주된 내용은 SMT 해결기[1]인 Z3 에 새로운 Theory 를 어떻게 추가하는지에 관한 내용이다. 새로운 Theory 는 다음과 같은 상황에서 필요할 수 있다. 예를 들어 사용자가 사용자 자료구조를 정의하고 그것을 이용하여 프로그램을 짠 후

[1] SMT 해결기에 관한 기초적인 내용은 http://rosaec.snu.ac.kr/trip/data/wslee_ropas.snu.ac.kr_20110612.pdf 의 SMT Theory and DPLL(T)에 관한 내용을 참조해주세요.

자료구조와 관련된 성질 검사를 SMT 해결기를 이용하여 하길 원하는 상황이다. 이 때 기존에 존재하는 Theory 를 이용하여 새 Theory 를 Encoding 할 수 있다. 우선 새로운 Theory 가 가져야 할 공리들을 표현하고, 이를 기존 Theory 의 Formula 로 바꾸어 쓴다. 그 후 주어진 새로운 Theory formula 도 기존의 Theory 의 formula 로 표현하고, 이 식에 앞서 써놓은 공리를 덧붙여 기존 Theory solver 로 푼다. Z3 는 느긋한 해결법(Lazy approach)을 취하고 있기 때문에 SMT 해결 과정에서 Theory solver 가 알게 된 사실을 풀고 있는 식에 추가하는 일이 자주 일어난다. 그리고 이를 위한 인터페이스 함수도 제공한다. 이 함수를 이용하여 비교적 간단하게 새로운 Theory solver 를 구현할 수 있다고 한다. 이는 물론 아예 새로운 Theory solver 를 구현하는 것보다 부담이 적다.

발표자는 클래스 상속에 관련된 Theory, 가중치 있는 MaxSMT (Weighted MaxSMT) 문제, 왼쪽 서브트리가 immutable 한 이진 탐색 트리에 관련된 Theory 를 기존 Theory(배열, 재귀적 자료형)들을 이용하여 어떻게 표현하는지 설명하였다. 이 때 각각의 새로운 Theory 를 Encoding 하는 방법이 여러가지가 있을 수 있는데, 어떤 방법을 택하느냐에 따라 성능도 달라지는 것이 신기했다.

THE TEACHING TOOL CALCCHECK: A PROOF-CHECKER FOR GRIES AND SCHNEIDER'S "LOGICAL APPROACH TO DISCRETE MATH"

이 논문의 주된 내용은 이산수학 교재의 증명문제들의 올바름을 체크해주는 증명 도구를 만들었다는 내용이다. 학생이 Latex 로 이산수학 증명을 작성하면(물론 CalcCheck package 를 사용하여 특정 command 들을 이용하여 써야한다) 증명이 올바른지 CalcCheck 이 판별해준다. Coq 이나 Isabelle 같이 배우기 쉽지 않은 도구들을 사용하기 보다는, Latex 만 사용할 줄 알면 이 도구를 사용할 수 있기 때문에 저학년 학부생들에겐 훨씬 실용적이다. 이 도구는 Haskell 로 (약 5 천여줄) 짜여졌다고 한다.

사실 이 논문은 내용만 두고 보자면 지적으로 흥미롭고 새로운 것은 많지 않다고 볼 수 있는 것 같다. 그럼에도 불구하고 이 논문이 훌륭하게 느껴지는 이유는 생활 주변에서 정말 필요한 문제에 대한 해결했기 때문이 아닐까 싶었다. 그러나 대개 논문을 승인받기 위해 요구되는 지적 흥미로움과 새로움이, 주변 문제에 대한 해결과정에서 늘 따라오지는 않는 것 같다. 그리고 그 경우 좋은 일을 하고도 논문 출판이 어려울 수 있다.

연구 본연의 목적이 주변 문제의 해결이고자 한다면, 이 논문과 같이 지적으로 새롭고 아름다운 이론에 바탕을 둔 연구가 아니라고 하더라도, 사람들에게 널리 알려질 수 있는 창구가 많아야 하지 않을까 하는 생각이 들었다. 더불어 그런 점에서 CPP 는 좋은 학회라는 생각을 했다.

느낀점

이번 학회는 좋은 발표란 무엇인지에 대해서 많이 생각해 볼 수 있는 기회가 되었다. 나는 내년 1 월에 있을 VMCAI 학회에서의 발표를 앞두고 있어서 이번 학회 기간 동안 주로 발표자들의 발표에 대해서 유심히 보았다. 그리고 발표에 대한 황금률은 앞선 페이지에서 언급했듯이 다음 두 가지라고 생각했다. 1. 발표자가 재미있어야 하고 2. 무엇을 어떻게 한 것인지 도입부에 명확하고 알기 쉽게 제시하는 것이다. 같은 연구실의 학주 형의 발표(Access based localization with bypassing)가 이 두 가지를 충족시켰다고 생각한다. 둘 다 만족이 되면 설령 기술적 세부사항이 좀 어렵게 설명되거나 슬라이드 구조가 전반적으로 어수선해도 청중들이 괴로워하면서 따라오는 걸 보았다. 그리고 따라오지 못한다고 해도 발표자에게 물어보거나 논문을 찾아서 읽어보게 된다. 그리고 설령 두 가지를 다 충족시키기 어려우면, 첫 번째는 충족시켜야 한다. 논문의 기여를 도입부에서 명확하고 이해하기 쉽게 제시하지 못한다고 해도 발표자가 발표하면서 재미있어 하면, 청중들은 무엇이 그리 재미있나 하고 호기심을 갖고 듣는 것 또한 목격했다. 위에서 소개한 연구 중에 The teaching tool CALCCHECK 에 대한 발표가 그렇다. 이 논문 발표는 사실 짜임새 있지는 않았다. 그러나 발표자가 느끼는 순수한 열정과 재미가 느껴지는 발표였다. 그래서 청중들도 관심있게 들었던 것으로 기억한다. 반면 좋은 연구를 한 것 같기는 한데 발표자도 재미없어하고 발표도 간결 명확하지 않아서 사람들의 관심을 받지 못하는 안타까운 경우들도 목격했다.

좋은 전달을 위해선 자신이 정말 의미 있는 일을 했다는 자부심과, 전달할 내용을 정제하여 최대한 간결하게 전달할 능력 두 가지가 필요하다고 생각했다. 전자는 연구에 대한 태도 문제이고, 후자는 문제에 대해 얼마나 많이 고민했느냐에 대한 문제일 것이다.

마치며

유익하고 즐거운 경험을 할 수 있게 해주신 이광근 교수님께 감사 드립니다


福華渡假飯店
Howard Beach Resort
KENTING
APLAS