CAV 2012

Berkeley, CA, US, Jul. 9th - Jul. 13th

소프트웨어 무결점 연구센터 이원찬

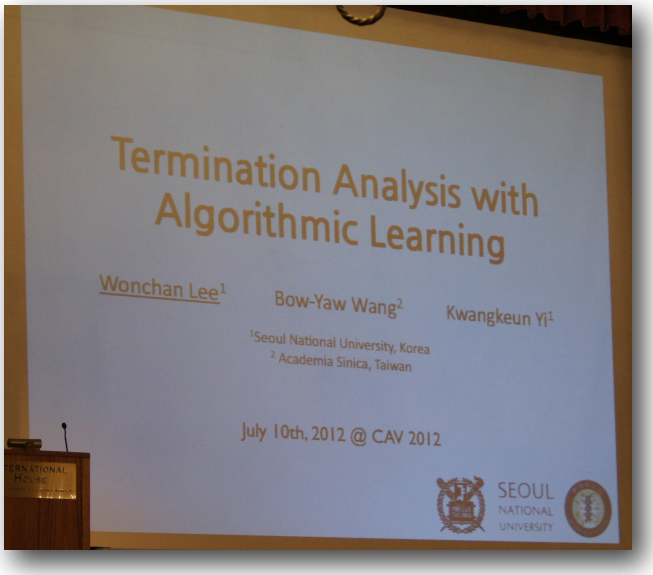
CAV 학회

CAV 학회는 자동 검증 분야의 최고 학회다. 올해 CAV는 정말 우연히도 방문 연구를 와 있는 이곳 버클리에서 열렸다. 게다가 이번에는 처음으로 학회에서 발표를 하는 기회를 가지게 됐다.

CAV 학회는 참 즐거운 학회였다. 지난 POPL 학회처럼 매 발표에서 배우고 감탄하는 시간은 물론 아니었다. 자동 검증 분야 학회인 탓인지 많은 발표는 내 관심 밖이거나 전혀 알아들을 수 없는 것들이었다. 그럼에도 즐거웠던 이유는 많은 사람들을 발표자로서 만날수 있었기 때문이다. 무엇보다 CAV 학회에 오는 사람들은 사뭇 분위기가 달랐다. 모두가 대화를 시작하는데 매우 열린 태도를 보였다. 무언가 설명할 수 없는 장벽에 압도되어 말도 못걸기 일쑤였던 지난 학회들과는 다른 경험이었다. 생각보다 많은 사람들이 내 발표에 관심을 가져주었고 말을 걸어왔다. 발표가 아주 나쁘지는 않았구나 하는 생각에 안도감이 들었다. 게다가 이번에는 아는 친구들과 함께였다. 원태가 있었고 같이 일하는 Vijay가 있었다. 정말 오랜만에 왕교수님을 다시 뵀고, 지난 여름 학교때 만났던 친구들도 함께 했다. 정말 즐거운 경험이었다. 학회가 단지 연구를 나누는 장을 넘어 친구들을 다시 만나는 장소가 되다니. 학회가 한층 가까워진 느낌이다.

발표들은 재밌으면서도 참 솔직했다. CAV 발표들은 목표가 꽤 구체적이다. 대체로 이런식이다. 알려진 문제가 있고 알려진 해법들이 있고, 하지만 이러저러한 이유로 좀 부족해서 내가 이런 새로운 아이디어를 시도해봤다. 새로운 아이디어는 모든 경우에 항상 잘되는 건 아닌데 해본 것들 중에는 잘되는 경우가 많았다는 식이다. 야심찬 발표가 아니었음에도 충분히 중요한 연구라는 인상을 새겨주었다. 이런 솔직한 연구들이 쌓여서 정말 가슴뛰는 연구가 나오는 것이 아닐까. 학자다운 정직함이 참 마음에 들었다.

발표 이야기



이번 CAV에서는 CDNF 학습 알고리즘을 사용하여 종료 분석을 하는 새로운 방법에 대해 발표를 하게 되었다. 지난 2년간 우리 연구실과 함께 CDNF 학습 알고리즘을 사용한 분석을 연구해오신 왕교수님과, 이광근 교수님이 함께한 연구다. 종료 분석은 프로그램이 항상 종료하는지를 증명하는 분석이다. 핵심은 반복문에서 일어나는 모든 변화를 포섭하는 이행 불변식을 찾고 그 이행 불변식이 무한한 실행을 표현할 수 없음을 보이는 것이다. 우리는 CDNF 학습 알고리즘을 사용하여 이행 불변식을 찾는데 이는 기존에 없던 방식이다. CDNF 학습 알고리즘은 대상 식에 대한 질문을 하고 그에 대한 답을 사용하여 대상을 적극적으로 배워나가는 학습 알고리즘이다. 질문에 답을 해주는 선생님을 적절히 만들어주면 학습 알고리즘은 우리가 원하는 대상 식을 찾아주게 된다. 우리가 제안한 방법에서 알고리즘의 역할은 어떻게 주어진 부품식을 조립하면 이행 불변식이 되는지를 알아내는 것이다. 이 과정에서 특정 부품식의 조합이 불변식에 포함되는지 혹은 우리가 찾는 이행 불변식인지를 묻는다. 기존에 있던 SMT 풀이기와 종료 검사기를 사용하여 이 질문들에 대한 대답을 해줄 수 있었다. 우리 방법이 기존의 방법들에 비해 가지는 장점은 이행 불변식이 주어진 부품식을 조립해서 만들 수 있기만 하면 찾아준다는 점이다. 기존의 방법은 이행 불변식을 빠르게 찾기 위해 탐색하는 식의 모양을 제한하기 때문에 종료하는 프로그램임에도 증명을 못하는 경우가 있다. 기존의 방법들과 윈도우 장치 드라이버 예제를 포함한 벤치마크를 사용하여 성능을 비교하였는데 기존의 방법에 비해 더 빨리 더 많은 프로그램의 종료를 증명할 수 있었다.

학회 발표를 준비하는 것도 처음이지만 이걸 방문연구 나와서 하게될 줄은 꿈에도 몰랐다. 연구실 사람들의 날카로운 지적들이 모여 단단한 발표가 되는 과정을 자주 지켜본 터라 그 든든한 지원이 아쉬웠다. 밖에 나와있다고 어설픈 발표를 하기 싫어 열심히 준비했다.

우선, UC 버클리 교수인 Sanjit Seshia에게 리허설 발표를 하고 싶다는 메일을 보냈다. Sanjit은 흔쾌히 나를 초대해주었고 CAV 학회 한 주전에 리허설을 할 수 있었다. 리허설 날짜가 잡히고 나는 독립기념일도 반납하며 발표를 준비했다. 원태네 그룹 미팅에 초대되어 같은 내용을 발표했던 경험이 도움이 됐다. 그때 만들었던 슬라이드를 바탕으로 20분 발표를 준비했다. 스크립

트는 만들지 않았다. 외우지 않아도 20분안에 발표를 할 수 있는지 알고 싶었기 때문이다. 슬라이드를 만들 때 주안점들을 기억하며 리허설에 임했다. 같이 일하는 Vijay도 와서 리허설을 들어주었다.

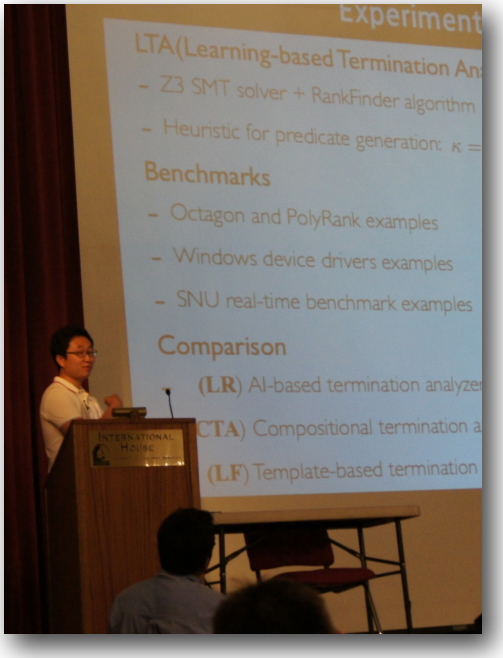
Sanjit의 그룹은 가족적이었다. Sanjit은 나를 일일이 소개해주었고 모두들 나를 반갑게 맞아주는 듯 했다. 그룹 사람들이 서로 즐겁게 이야기 나누는 모습을 보니 연구실이 생각나 돌아가고픈 마음이 들었다.

리허설은 본 발표 준비에 많은 도움이 됐다. 우선, 스크립트가 꼭 필요하다는 사실을 깨달았다. 스크립트가 없어서 20분안에 마치긴 했지만 몇몇 부분을 빼먹고 넘어갔다. Sanjit의 꼼꼼한 질문들도 도움이 됐다. Sanjit은 진심으로 내 발표를 이해하려고 노력했고 이해가 가지 않는 부분을 잘 적두었다가 내게 질문했다. 발표 후에 슬라이드를 한 장씩 넘겨가며 같이 검토했다. 그리고 격려하는 말도 아끼지 않았다. Vijay도 좋은 이야기를 많이 해주었다. 전반적인 발표 방법에 대한 이야기들이었다. 기억에 남는 것들을 공유해보면 다음과 같다. '발표를 한 장 한 장 끊어서 하기 보다는 이야기를 이어나가듯 하는 것이 좋다.' '슬라이드 없이도 혼자서 줄거리를 말할 수 있어야 한다.' '발표 듣는 사람을 구체적으로 한정하는 것이 좋다. 내 발표와 비슷한 연구를 한 사람들이지만 이해의 정도가 다른 세 사람 정도를 가정하고 준비하는 것이 좋다.' 마지막 조언은 조금 생소했는데 생각해보면 수긍이 되는 부분이었다. 물론 모든 사람이 다 이해할 수 있는 발표가 가장 좋은 발표일 것이다. 하지만, 학회 발표고 게다가 비교적 좁은 분야를 다루는 CAV 학회다. 그만큼 발표 듣는 사람의 지식 수준을 좀 더 고려해야 지루하지 않은 발표를 할 수 있겠다는 생각이 들었다.

리허설 때 받은 질문들과 연구실 사람들로부터 받은 조언들을 바탕으로 발표를 수정했다. 그리고 스토리를 만들고 스크립트를 준비했다. 스크립트는 몇 번이나 소리내어 읽어보고 어색하면 고치기를 수 차례 반복했다. 이렇게 완성된 스크립트를 외울 차례가 됐는데 당췌 외워지지가 않았다. 그래서 묘안을 생각해냈다. 각 슬라이드마다 딱 세 문장만을 설명하게 만드는 것이다. 만일 한 슬라이드를 가지고 세 문장 이상 말해야 하면 이를 둘 이상로 나누었다. 이렇게 했더니 뇌에 부담이 적었던 탓인지 좀 더 잘 기억할 수 있었고 발표 때도 많이 떨리지 않았다. 무엇보다 한 번에 한 주제만 말을 하게 돼어 헷갈리지 않았고 명확히 설명할 수 있었다. 프리젠테이션 화면과는 별도로 발표 시간과 스크립트를 다른 쪽 화면에 보여주는 키노트의 기능이 요긴하게 쓰였다.

발표장에 들어섰다. 내 발표가 속한 세션은 전체가 학습 알고리즘 혹은 종료 분석에 대한 발표로 구성됐다. 첫 발표는 왕교수님이 학습 알고리즘을 확장한 내용에 대해 이야기 하셨다. 우리가 쓰는 학습 알고리즘인 CDNF 알고리즘은 고정된 갯수의 Boolean 변수에 대한 식을 알아낸다. 왕교수님 발표는 어떻게 하면 적은 수의 변수에서 시작해서 필요할 때만 변수를 늘려나가면서 식을 찾는 방법에 대한 것이었다. 핵심은 학습을 통해 알아낸 정보를 변수가 늘어나고 나서 어떻게 재사용할지다. 흥미로운 내용이었지만 내 발표에 신경이 쓰여 집중해서 듣지 못해 죄송한 마음이 들었다. 이윽고 했는지도 모르게 두 번째 발표가 끝이 났고 내 차례가 돌아왔다. 발표하는 단상이 참 높았는데 거기 서서 내려다보니 낯익은 얼굴들이 보여 연구실 세미나를 하는 느낌이 났다. 발표를 시작하니 원태의 지도교수님인 Koushik 교수님이 막 들어오는 모습이 보였다. 눈이 마주치고 서로 씩 웃었다. 스크립트를 화면에 띄워놓은 탓인지 마음이 참 든든했다. 세 마디씩, 내가 하려고 마음 먹었던 말들, 전하고 싶은 메시지를 차근차근 전달했다. 별로 긴장이 되지 않다가 종반에 가서 가슴이 뛰었는데, 이유는 화면에 있어야 할 글이 사라져버렸기 때

문이다. 아마도 마지막으로 자료를 손대면서 지워져 버렸나보다. 쿵쾅거리는 심장을 쓸어담고 원래 화면에 있었어야 할 말을 했다. 사람들이 알아들었는지 어쨌는지 표정들을 봐서는 알 길이 없다.



발표는 20분 안에 무사히 끝이 났고 두 개의 질문을 받았다. 하나는 체어였던 Sriram Sankaranarayanan이 이행 불변식을 찾는데 쓰는 부품식을 어떻게 만드는지 물었다. 실험 부분 설명하면서 언급했는데 부족했었나보다. 슬라이드를 앞으로 돌려 우리가 사용한 휴리스틱에 대해 설명해줬다. 다른 하나는 Isil Dillig이 언제 거짓 경보가 생기는 지를 물었다. 기본적으로 종료 증명을 실패하는 경우 모르겠다고 답하기 때문이 이 경우에 거짓 경보가 생긴다고 답했다. 실제로, 여러가지 이유로 모르겠다고 답할 수 있는데, 부품식이 부족하거나 사용하는 종료 검사기 알고리즘이 완전하지 않거나하는 경우가 그 원인들이다. 나중에 발표장을 나와서 Isil이랑 잠깐 인사했는데 내게 좋은 발표였다며 말을 건넸다. 갑자기 미안한 마음이 들었다. 나는 Isil의 지난 PLDI 발표를 보면서 이상하다고 생각했기 때문이다. 주위 사람들에게 그 발표에 대해 혹평했던 기억이 떠올라 손발이 오그라들었다.

발표 끝나고 Nishant라는 사람이 내게 말을 걸었다. Nishant도 학습 알고리즘을 사용한 검증에 대해 연구한 적이 있었다고 한다. 발표 내용에 대한 이런 저런 질문을 하고는 재밌는 이야기를 꺼냈다. 자기가 학회에서 만난 한국인들은 우리 이광근 교수님을 알고 있거나 교수님의 제자거나 교수님과 같이 일한 적이 있었다고 한다. 그러고는 우스개소리로 자기 생각에 한국 연구자들 마피아가 있는 것 같다며 너스레를 떨었다. 내게 그 중 일원이냐고 묻기도 했다. 그는 우리 연구실의 업적을 꽤 소상히 알고 있었는데 분석기를 상용화한 Sparrow 이야기도 알고 있었다. 외국에 나와 이광근 교수님과 선배님들이 이룬 업적에 대한 이야기를 듣노라니 내가 한일도 아닌데 뿌듯하고 자랑스러웠다. (나중에 돌아와 검색해보니 Nishant은 Edmund Clarke교수님의 박사 졸업생이었다)

기억에 남는 발표들

IC3 and Friends: Incremental, Inductive Verification

Aaron R. Bradley

요즘 화두가 되고 있는 모델 검증 알고리즘인 IC3에 대한 튜토리얼 시간이었다. 혹자는 Ken McMillan이 만든 내삽(interpolation)을 사용한 모델 검증 알고리즘이 나온 이래 처음으로 진정한 알고리즘의 개선이 이루어졌다는 평을 한다. 아이러니한 것은 이렇게나 회자되는 알고리즘이 발표되는데 우여곡절을 겪었다는 점이다. 이 알고리즘이 2년 전 CAV 학회에 발표되었을 때는 아무도 제대로 이해한 사람이 없어서 믿을 수 없다는 반응이 지배적이었다고 한다. 이에 분개한 Aaron은 새 알고리즘을 가지고 하드웨어 모델 검증기를 만들어 2011년 모델 검증 대회에 출전했고 3위에 입상했다. 이 3위는 굉장히 놀라운 결과인데 이유는 이렇다. 모델 검증 대회

에 출전한 대부분의 검증기들은 1) C로 구현되었고 2) 성숙한 BDD 라이브러리를 사용하며 3) 온갖 휴리스틱과 최적화로 무장한 상태였다. 하지만, Aaron이 만든 검증기는 OCaml과 SAT 풀이기를 사용해서 만들었다. OCaml과 C의 성능차는 말할 것도 없고 SAT 풀이기를 쓰는 것보다 BDD를 쓰는 편이 성능이 더 낫다는 것은 알려진 사실이다. 비록 3위를 했다고 하나 사실상 알고리즘 자체는 기존 알고리즘을 압도했다고 보는 것이 맞을 것이다. 이 결과는 그 해 VMCAI 학회에 발표되었으며, 그 이후로 검증 분야의 많은 사람들이 IC3를 이해하고 구현하기 위해 달려들었다고 한다. 실제로 2012년 모델 검증 대회에서 상위에 입상한 검증기들은 대부분 IC3 알고리즘을 구현하고 있다.

튜토리얼은 IC3를 바닥까지 내려가서 설명하기 보다는 왜 IC3가 잘 동작하는지, 기존의 방법과는 어떻게 다른지를 주로 다루었다. 가장 큰 차이점은 기존의 방법은 하나의 커다란 귀납 불변식을 찾는데 비해 IC3는 서로에 대해 귀납적인 작은 불변식 여러개를 찾는다는 것이다. 구체적인 차이는 다음과 같다. 검증은 이행 관계가 T인 시스템이 어떤 성질 P를 항상 만족하는지 보이는 것이다. 만일 성질 P가 이행 관계 T에 대해 귀납이 성립하는 경우($P \wedge T \Rightarrow P'$), P 자신이 시스템의 불변식이므로 시스템은 P를 만족한다. 대부분 P는 귀납이 성립하지 않는 일반적인 식이기 때문에 P에 포섭되지만 귀납이 성립하는 다른 불변식 F를 찾는다. IC3 이전의 대부분의 검증 알고리즘은 F를 곧바로 찾으려 시도한다. 허나 IC3는 일련의 F들을 한꺼번에 찾는다. 각 F_i는 F_{i+1}에 포섭되며 F_{i+1}은 F_i에서 한 번 더 실행된 상태들도 포섭한다. 그리고 가장 마지막 Fi를 제외하고는 모두 성질 P를 만족한다. 만일 이런 F들 가운데 $F_i = F_{i+1}$인 F_i가 있으면 귀납 불변식을 찾은 것이 된다. IC3가 기존에 비해 잘 동작하는 이유는 내 생각에 두 가지이다. 크게는 Fi가 i번 실행한 상태들의 요약이라는 점을 활용할 수 있다는 것이다. 만일 어떤 F_i에서 귀납을 만족하지 않는 상태 s가 발견될 경우($F_i \wedge s \wedge T \not\Rightarrow F_{i+1}$), F_i에서 부터 F_1에 이르기 까지 역으로 쫓아가며 s에 도달할 수 있는 모든 경로를 쉽게 없앨수 있다. 모든 실행을 한꺼번에 포섭하고 있는 F 하나만 관리하는 경우 이같은 방법을 쓰기가 쉽지 않다. 또한, F를 여러 F_i들로 쪼개어 놓은 덕분에 식의 크기가 줄어들어 SAT 풀이에 드는 비용도 줄어드는 것도 IC3가 좋은 성능을 보이는 이유가 된다.

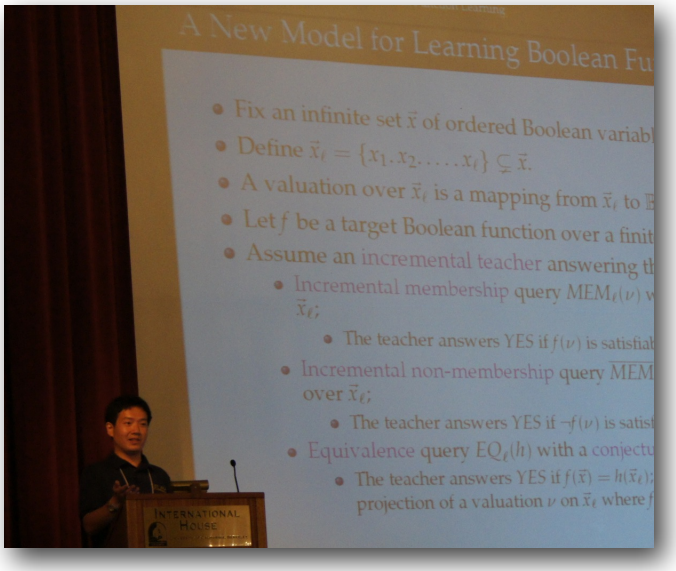
비록 한 시간이 넘는 튜토리얼을 들었지만 아직 제대로 이해한 것 같은 확신이 들지 않는다. 더 공부해봐야 겠다는 생각이 든다. 유행을 굳이 좇을 필요는 없지만 IC3가 불변식을 찾는 과정을 공부하다보면 우리가 하는 분석에도 도움이 될 것 같다는 생각이 든다. 또, 한 가지 IC3를 공부하고 싶은 이유는 내가 계속 공부해오던 CDNF 알고리즘과 닮은 구석이 있어서이다. 두 알고리즘 모두 어떤 식 F이 상태 s를 포함하지 않게 하는 방법으로 $\neg s$의 일부와 F와의 교집합을 취하는 방법을 사용한다. 내가 모르는 저 너머에 두 알고리즘 사이의 깊은 관계가 숨어있을것 같다는 생각이 든다.

Learning Boolean Functions Incrementally

Yu-Fang Chen and Bow-Yaw Wang

변수 갯수를 늘려가며 Boolean 함수를 배우는 CDNF 알고리즘을 소개하는 발표였다. 원래 CDNF 알고리즘은 주어진 Boolean 함수를 알고리즘도 다항시간안에 "배울 수 있는지"를 연구하는 과정에서 얻어진 산물이다. 그래서 함수에 나타난 Boolean 변수의 갯수는 고정이 돼 있다. 이런 가정에서는 변수의 갯수를 늘려가며 대상 함수를 배우는 알고리즘은 의미가 없다. 이미 대상 함수를 포현하기 위해 몇 개의 변수가 필요한지 주어져 있기 때문이다. 하지만, 최근 왕교수

님과 우리 연구실에서 CDNF 알고리즘을 분석 문제에 적용하면서 가정이 바뀌었다. 분석에서 CDNF 알고리즘을 사용하는 목적은 우리도 모르는 대상 함수를 배우는데 있다. 정확히는 주어진 부품식의 Boolean 조합을 CDNF 알고리즘으로 하여금 알아내게끔 하는 것이다. 하지만, 우리가 넘겨준 부품식 중 일부는 불변식을 표현하는데 필요하지 않을 수도 있다. 이런 경우 CDNF 알고리즘이 필요 이상으로 많은 수의 Boolean 변수를 사용하여 대상을 표현하려 들기 때문에 비효율적이다. 발표에서는 이를 개선하기 위해 적은 수의 변수에서 출발하여 주어진 변수로 대상을 표현할 수 없는 경우에 그 갯수를 늘려나가는 알고리즘을 제안했다.



변수 갯수를 늘려가며 배우기 위해서 풀어야 하는 숙제는 언제 변수 갯수를 늘려야 하는지 알아내는 것이다. 아이디어는 간단하다. 지금 주어진 변수가 대상을 표현하기에 충분하다면 절대로 일어날 수 없는 일을 감지하는 것이다. 구체적으로는 이렇다. 대상을 주어진 Boolean 변수들의 진리값 u로 치환했을때 참이 될 수 있는지, 참이 될 수 없는지를 알려주는 "점쟁이" MEM(u), NOTMEM(u)가 있다고 하자. 만일, 알고리즘이 충분한 수의 Boolean 변수에 대해 진리값을 준다면 MEM이 "네"라고 대답할 때 NOTMEM은 반드시 "아니오"라고 대답해야 한다. 다시 말해, 두 점쟁이가 같은 대답을 하는 경우가 있다면 이는 충분한 수의 변수를 사용하고 있지 않다는 말이 된다. 예를 들어, 대상 함수가 $b_1 \wedge b_2$라고 하자. 만일, u가 {b_1: T, b_2: T}라면 MEM(u)는 "네"라고 대답하고 NOTMEM(u)는 "아니오"라고 대답할 것이다. 하지만, u가 {b_1: T}라면 두 점쟁이 모두 "네"라고 대답할 것이다. 주어진 u가 b_2의 값을 정해주지 않아서 참이 될 수도 있고 거짓이 될 수도 있기 때문이다.

논문에서는 변수 갯수를 늘려야 하는 시점을 알아내는 아이디어를 가지고 두 가지 알고리즘 CDNF+와 CDNF++을 만들었다. CDNF+의 경우 변수의 갯수를 추가하고 나서 처음부터 다시 배우는 반면 CDNF++는 이전까지 배운 내용을 추가된 변수에 대해 확장한다. 실험을 통해 이 둘의 성능을 기존의 CDNF 알고리즘의 성능과 비교했다. 비교에는 지난 반복문 불변식 추론 논문때 썼던 구현을 사용했다. 두 알고리즘 모두 최악의 경우에는 기존의 CDNF 알고리즘과 비슷한 성능을 보였다. 하지만, 추가하는 변수의 순서를 잘 지정하는 경우 두 알고리즘 모두 CDNF 알고리즘에 비해 많이 개선된 결과를 보여주었다. 재밌는 점은 CDNF+가 CDNF++에 비해 더 나은 성능을 보였다는 것인데, 이는 배운 내용을 확장하는데 필요한 계산이 처음부터 다시하는 것보다 많기 때문으로 보인다.

The Gauge Domain: Scalable Analysis of Linear Inequality Invariants

Arnaud Venet

선형 부등식으로 된 불변식을 표현하는 새로운 domain을 소개한 발표였다. 배열 접근이 안전함을 증명하기 위해서는 변수가 가지는 숫자 값을 분석해야 한다. 그동안 제안된 domain들은 효율적이지만 표현력이 떨어지거나(예를 들어, interval domain 이나 octagon domain), 표현력

이 좋지만 매우 비효율적이다(예를 들어, polyhedra domain). NASA에서 이들 domain을 가지고 분석을 해보려 했으나 성능이 좋지 않거나 정확하지 않아서 새 domain을 만들었다고 한다.

Gauge domain은 변수가 갖는 값의 범위를 반복문이 반복되는 횟수에 선형인 식으로 표현한다. NASA에서 쓰는 코드에서 자주 발견되는 패턴이라고 한다. 예를 들어, `x=0; while(*) { if(*) x++; else x+=2; }`와 같은 프로그램에서 x가 갖는 값은 반복문이 반복되는 횟수를 λ이라 할 때 $\lambda \leq x \leq 2\lambda$로 표현할 수 있다. 이를 임의로 중첩된 반복문에 대해 적용하면 $[l_0 + l_1\lambda_1 + l_2\lambda_2 + ..., u_0 + u_1\lambda_1 + u_2\lambda_2 + ...]$와 같이 표현할 수 있다. 널리 쓰이는 interval을 반복문 카운터마다 따로 가지고 있는 domain으로 이해하면 된다. 합치기와 순서 또한 해당되는 interval의 연산을 각 계수들에 따로 적용하는 식으로 확장한 것이된다(사실 이렇게 되는 이유는 자명하지 않은데, 자세한 내용은 논문을 참조). 각 반복문 카운터에 붙은 계수는 내삽으로 구할 수 있으며, 이는 곧 축지법 연산이 된다. 이렇게 설계된 domain으로 NASA에서 쓰는 14만 라인 크기의 프로그램을 분석했더니 놀랍게도 interval domain보다 빠르면서 상용 정적 분석기만큼 정확한 결과를 얻었다고 한다. 물론 분석 대상에 대한 지식을 바탕으로 만든 domain이기 때문에 이런 결과를 얻을 수 있었겠지만, 매우 효율적이라고 알려진 interval domain보다도 빠르다는건 정말 충격이었다. 연구실에서 UAV 코드를 분석하는 친구들이 있다고 들었는데 이 gauge domain을 소개해주고 싶었다.

Arnaud Venet는 논문을 자주 발표하는 분은 아니지만 매번 놀라운 결과를 들려주시는 것 같다. 그만큼 훌륭한 분이시기도 하겠지만, 진짜 문제에 가까운 곳에서 깊게 파고들기 때문에 이런 결과들이 나오는 것 아닐까. 나도 곁눈질 그만하고 한 우물을 깊게 파야겠다는 반성을 하게 되었다.

Efficient Runtime Policy Enforcement Using Counterexample-Guided Abstraction Refinement

Matt Fredrikson, Richard Joiner, Somesh Jha, Thomas Reps, Phillip Porras, Hassen Saidi and Vinod Yegneswaran

분석을 사용해서 보안 정책을 어길만한 부분에만 감시자 코드를 집어넣는 방법을 소개한 발표였다. 프로그램이 보안 정책을 준수하도록 강제하기 위해 곳곳에 삽입한 감시자 코드를 IRM(In-lined Reference Monitor)라 부른다. 만일 보안 정책을 어기는 실행 흐름이 감시자 코드에 도달하면 코드는 강제로 프로그램을 종료시킨다. 여기서, 감시자 코드는 실행 시간에 정책의 위반 여부를 검사하므로 프로그램의 성능을 떨어뜨릴 수 있다. 따라서, 감시자 코드를 필요한 부분에만 집어넣어야 과다한 성능저하를 막을 수 있다.

이 발표에서는 감시자 코드를 넣을 필요가 있는 부분을 CEGAR(CounterExample-Guided Abstraction Refinement)를 사용해서 찾는다. CEGAR는 성긴 요약에서 시작해서 반례를 통해 검증에 필요한 만큼 요약을 정확하게 만들어나가는 방법이다. 반례는 주어진 요약에 포함된 상태 중 검증하고자 하는 성질을 만족하지 않는 예를 말한다. 요약이 너무 성기기 때문에 생긴 반례를 허위 반례라고 한다. 허위 반례를 찾았다면 같은 이유로 그 반례가 다시 생기지 않게끔 요약을 정확하게 만든다. 만일 반례가 실제로도 생길 수 있는 경우라면 오류를 찾은 것이다. CEGAR는 이 같은 반례 찾기와 요약 정확하게 만들기를 실제 오류를 찾거나 반례를 더이상 찾을 수 없을 때까지 반복하기 때문에 정확하지만 시간이 오래 걸린다. 발표에서는 절충점을 제안한다. 실제 반례를 찾은 경우, 보안 정책을 실제로 어기는 경우이므로 감시자 코드를 집어넣는

다. 만일, 허위 반례인 경우는 두 가지로 대처한다. 요약의 정확도가 일정 수준 이하라면 요약을 정확하게 만든다. 만일 요약이 어느 수준 이상으로 정확하다면 이후 발생되는 모든 허위 반례에 대해 감시자 코드를 삽입한다. 발표에서는 상태를 부품식으로 요약하는 방법(predicate abstraction)을 사용하는데, 추가할 부품식의 갯수에 제한을 둠으로써 요약을 정확하게 만드는 정도를 조절할 수 있다. CEGAR를 수행하는데 드는 시간을 절약하는 대신 필요없는 감시자 코드를 집어넣는 것이다. 물론, 요약을 정확하게 만드는 정도를 조절함으로써 필요없는 감시자 코드를 충분히 줄일 수 있다. 실제 Javascript 프로그램들에 대한 실험 결과에서도 필요없는 감시자 코드의 수가 한 개 내외로 작았다.

재미있는 조합이었다. CEGAR가 프로그램 전체를 검증하기까지 시간이 오래 걸리는 단점을 IRM을 통해 "우아하게" 처리한 셈이다. 오랜 시간동안 분석하고 그냥 잘 모르겠다고 하기 보다는 적어도 오류를 방지하는 코드를 집어 넣으니 말이다. 물론, 검증 대상이 보안 정책이 아니라 버퍼 오버플로우 같이 검사해야 할 곳이 더 많은 속성이었다면 잘 동작하지 않았을것 같다. 버퍼 오버플로우 분석에 대해 같은 아이디어를 적용했다면 아마 대부분의 배열 접근에 감시자 코드를 집어넣어야 했을 것이다.

Automatic Quantification of Cache Side-Channels

Boris Köpf, Laurent Mauborgne and Martín Ochoa

캐시의 정적분석을 통해 프로그램이 캐시를 통한 공격에 얼마나 취약한지를 알아내는 방법을 다룬 논문이다. 몰랐던 사실인데 캐시 메모리로 인해 실행 시간이 달라지는 것을 재는 것만으로도 무려 암호화에 사용된 키 값을 알아내는 것이 가능하다고 한다. 이 같은 공격을 시간차 공격(timing attack)이라 하는데 실제로 AES나 RSA같이 널리 쓰이는 암호화 알고리즘에도 유효하다고 한다. 더 큰 문제는 암호화 알고리즘이 아닌 프로그램들도 이런 공격에 취약할 수 있음에도 얼마나 취약한지 알 방법이 그동안 없었다는 사실이다. 논문에서는 정적 분석으로 캐시 상태에 대한 안전한 요약을 사용하여 캐시를 통해 얼마나 많은 자료가 샐 수 있는지를 측정하는 방법을 소개했다. 캐시를 통해 유출될 수 있는 최대 정보량은 캐시가 가질 수 있는 상태의 엔트로피의 최대값이라고 한다. 캐시가 가질 수 있는 모든 상태들의 안전한 요약은 캐시를 정적분석하여 얻을 수 있다. 따라서, 각 상태들이 발견될 확률이 균일하다고 했을 때 엔트로피의 최대 값은 발견 가능한 캐시 상태 갯수에 로그를 취한 값이된다. 다행히도 정적 분석으로 얻어지는 캐시 상태의 요약은 유한하게 많은 캐시 상태를 표현하기 때문에 구체화하여 갯수를 세는 것으로 그 상한을 구할 수 있다. 논문에서는 엔트로피의 최대값을 좀 더 정확히 구하기 위해 각 캐시 라인에 존재할 수도 있는 값들을 구하는 분석("아마도" 분석)과 반드시 존재하는 값을 구한 분석("반드시" 분석)을 같이 사용했다. 이 방법으로 PolarSSL 라이브러리에 들어있는 AES 알고리즘 구현이 암호화 키를 얼마나 유출시킬 수 있는지 16KB와 128KB의 두 크기의 캐시에 대해 쟀다. 측정 결과 캐시 크기에 상관없이 최대 160비트를 유출시킬 수 있었는데, AES에 사용되는 키가 128비트인걸 감안하면 매우 취약하다고 할 수 있다. 재밌는 것은 AES 구현을 조금 고쳐서 암호화 하기 이전에 사용되는 배열들을 미리 캐시로 읽어들이도록 했더니 128KB 캐시를 사용하는 경우 단 1비트의 키 값만을 유출했다는 점이다. 이는 직관적으로도 당연한 결과인데, 계산에 쓰이는 배열들이 이미 캐시에 들어있는 상태이기 때문에 캐시 미스때문에 시간이 더 걸리더라도 어느 값을 새로 읽어들였는지가 "불확실"할 것이기 때문이다. 이런 종류의 공격을 막으려면 중

요한 계산을 하는 코드에서는 반드시 계산에 쓰는 모든 값들을 캐시로 먼저 읽어들일 필요가 있음을 시사하는 실험 결과였다.

정적분석의 재발견이었다. 개인적으로 가장 마음에 드는 발표다. 그동안 내 머리속에는 오류 검출이 목적인 분석밖에 들어있지 않았다. 분석으로 구하는 요약 의미는 프로그램이 하는 일을 들여다보는 훌륭한 도구다. 오류 검출만이 그 목적일 필요는 없을 것이다. 프로그램이 어떤 정보들을 네트워크로 실어 보낼 수 있는지, 어떤 권한을 사용하는지 같은 것들도 분석을 활용할 수 있는 멋진 예가 될 것이다. 무엇을 분석할 수 있을까를 좀 더 고민한다면 이 발표처럼 유용하면서도 우리가 가진 기술로 잘 할 수 있는 것들을 찾을 수 있지 않을까.

맺으며...

학회 발표를 들으며 다시금 한 주제를 집요하게 파고드는 끈기가 중요하다는 것을 깨달았다. 이해하기 쉬운 발표는 있었을지언정 그 결과에 이르는 과정이 당연해 보이는 발표 하나도 없었다. 지금 연구하고 있는 학습 알고리즘을 그 정수까지 파고들어야 겠다는 생각이 든다. 분명 고정점을 계산하는 새로운 방법에 대한 비밀이 숨어있을 것이라 믿는다.

끝으로, 함께 준비하고 연구했던 이광근 교수님과 왕교수님께 진심으로 감사를 드리고 싶다. 나는 논문 리뷰를 받았을 당시 점수가 너무 좋지 않아 반쯤 포기했었다. 하지만, 두 분 교수님께서 포기하지 않게끔 격려해주시고 답변 잘 작성하도록 도와주신 덕분에 좋은 잔치에서 발표를 할 수 있었던 것 같다. 논문과 발표자료를 살펴봐주고 많은 조언해준 연구실 여러분께도 감사를 드린다.