CS320 Programming Language

20130538 Taesoo Lim

Guest Lecture – Introduction to Automatic Software Verification by Hongseok Yang

지난 월요일에 동영상 강의 숙제만을 남기고 비어있었던 강의실의 기억을 뒤로한 채 들어선 터만 홀의 교단엔 낯이 익지 않은 실루엣의 한 남성이 류석영 교수님의 자리를 대신하고 있었다. 꾸밈없는 흰색 배경에 강의 제목과 본인의 이름만을 밝히는 파워포인트 슬라이드를 정면 화면에 띄워놓은 채, 앞자리에 앉아있던 나를 포함한 몇몇 학생들과 우리 대학에 대한 간단한 질문 몇 개를 주고 받은 그는 이내 썩 유창한 영어를 뽐내며 초청 강연을 시작하였다. 그의 참으로 아시아인다운 발음으로 이루어진 매끄러운 영어에 놀라던 것도 잠시, 정규 수업이 아니니 듣고 싶지 않으면 듣지 않아도 된다는 참으로 쿨한 그의 첫마디에 문득 지난 주 옥스포드에서 교수님 한 분을 모셔서 강연을 할 것이라는 류석영 교수님의 한마디가 스쳐 지나갔다. 물론 그렇다고 나를 포함한 대부분의 학생들이 강연을 듣고 싶어서 교실을 빠져나가지 않은 것은 아니고, 그냥 이거 듣고 관련 숙제가 나올 테니 어쩔 수 없이 그냥 거기 앉아 있었으리라. 이렇게 말하면 이 대목에서 “하지만 다 듣고 보니 역시 듣길 잘했다” 같은 맥락의 말이 나와야 이상적일 테지만, 애석하게도 상당히 미묘한 강연이라 기억될 것 같다. 강의 내용이 중간 지점까지 믿을 수 없을 정도로 쉽고 명백한 이야기로 나아가다가 중간에 잠깐 차원이 달라지듯 내용이 급속도로 난해해지는 시점이 있었다. 물론 이건 학생 쪽의 문제일 것이다. 교수님은 학생들이 모두 어딘가에서 내용을 놓친 듯 하다고 말씀하셨다. 하지만 단순히 그런 문제는 아닌 것 같다. 난 이 강의를 들으며 거의 모든 시간 집중했던 것 같지만 머릿속엔 양홍석 교수님의 유창한 영어밖에 남아있지 않다. 그렇기에 강연에서 배운 것과 함께 이 기묘한 느낌을 최대한 써보려 한다.

강연의 내용은 Software Verification이란 제목 그대로 어떻게 프로그램의 오류를 자동으로 찾을 수 있는지에 대한 내용이었다. 이런 생각을 맨 처음 제시한 것은 놀랍게도 앨런 튜링이었으며, 그의 1949년 6월 논문에는 에러 감지 자동화에 대한 상당히 현대적인 접근이 담겨 있다고 한다. 본 강연에서 다뤄졌던 부분은 추상화를 통해 코드의 이해를 하나도 하지 못한다는 가정 하에 초기값과 결과값만을 가지고 코드의 오류를 판별할 수 있는 기본적인 접근법에 대한 것이었다. 이를테면 두 수의 덧셈에서 홀수/짝수 배합에 따라 결과값이 홀수인지 짝수인지 예상할 수 있으니 만약 맞지 않으면 틀린 것이다. 혹은 5와 짝수를 곱했을 때 마지막 자리가 0으로 끝나는지 검사한다던가 하는 식의 접근법이다. 이를 software abstraction에 적용하기 위해 abstract value를 state의 값으로 가지는 리스트를 만드는데, 피보나치 수열을 예로 들면, 각 변수와 변수의 값을 튜플로 가지는 리스트 대신 양수(+)인지 수 전체(타우로 표시)인지만 표기하고 이를 boolean으로 핸들링 한다. 좋은 추상화를 위해 search problem의 접근법을 적용하여 verifier 모델을 만들고 search space를 지정하여 프루닝과 예측을 진행한다..... 사실 이쯤부터 굉장히 멍 하게 듣게 되었던 것 같다. 언젠가 류석영 교수님이 말씀하셨던 것처럼 분명 알아듣고 있다고 생각하며 들었는데 다 듣고 나니 내가 뭘 들었는지 생각이 나지 않는다.

이 강의에서 가장 중요했다고 생각되는 부분은 verifier가 코드의 이해를 하나도 하지 못한다는 가정에 있다고 생각한다. 이는 당연하다고 생각하며 강연을 들었는데 어느 순간 보니 피보나치 수

열 계산 iteration이 끝난 후에 왜 y가 양수인지를 증명할 방법이 없는지에 대해 질문을 하는 내가 있었다. Verifier는 y의 초기값이 1이고, while문 뒤에 y에서 1을 빼니 그 중간과정은 이해를 못한다고 하면 y가 양수인지 0인지를 확인할 방법이 없는 것이다. 강의 초반에 다뤄진 arithmetic expression의 abstraction부분과 이어지는 부분이었겠지만, 초반부에 이상하리만치 당연한 부분에 대해 대수롭지 않게 흘려 들었으면서 이어지는 것은 생각지 못한 것이 아닐까 싶다. 그런데 또 어려운 강의였냐고 물어본다면, 그렇다고도 말할 수 없을 것 같다. 사실 verifier가 코드의 이해를 아예 못한다는 것을 머리에 세기고 들으면, 그냥 pruning을 통해 y값이 0이 되지 않는 abstraction parameter들을 search space에서 골라낸 뒤 남은 것들을 넣어서 테스트 하고, 가장 reasonable한 parameter를 고르겠다는 것이다. 아마 내가 강연이 끝나고 나서도 뭔가 이해하지 못한 것 같은 느낌이 드는 이유는 "그래서 이걸 어디다 쓰지???" 라는 생각 때문일 것이다. 마치 PL을 공부하는 내내 이 과목의 존재 의의에 의문을 품었던 지난날의 나처럼, 강의를 듣는 내내 어디다 적용해야 하는지도 모를 abstraction parameter와 결국 끝까지 정체를 알아내지 못한 verifier에 대해 초기값 설정만 하고 적용은 보여주지 않는 강의를 보고서 그런 의문을 품었음이 당연했다. 정작 현실적인 수치가 나오는 것은 맨 뒷장의 한 페이지인데…… 그래프 하나로 작동 원리를 알 수는 없는 것이다.

소프트웨어 자동화 분석 쪽은 꽤 심심찮게 들려오던 강연 주제였다. 물론 내가 이 주제에 대해 들었던 곳이 대부분 정보보호대학원 세미나 포스터였고, 난 카이스트 해킹동아리 GoN의 부원이기 때문에 취약점 분석에 대해서만 알고 있었다. 이번에 들은 강연처럼 소프트웨어의 작동 방식 자체의 결함에 대한 분석은 사실 생각하지 않았던 부분이었다. 이번 강의로 접근법에 대해 느껴보길 기대했지만, 사실 아직 잘 모르겠다. 다년간의 대학생활로 깨달은 것 중 하나는, 이런 문제들은 보통 학생 쪽에서 수업 내용을 따라가지 못해서 생기는 경우라는 것이다. 아직 난 Software Verification 분야를 공부할 준비가 되지 않았다고 받아들이며, 공부가 부족함을 깨닫는 밤이 될 것 같다.

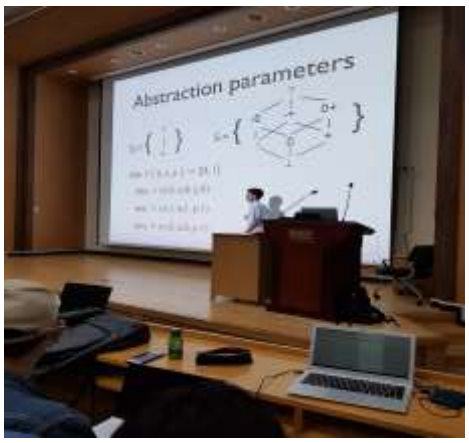

Figure 1 – 혹시 강의자료를 못 받을 때를 대비해 찍어둔 슬라이드 중 한 장면