이데아를 향한 노력

김재호

2022.09.07

Abstract

우리가 구현하는 모든 프로그램은 상상 속 이데아를 따라 만들어짐에도 결함이 발생할 수밖에 없다. 수학을 이용한 관념적 모델인 논리 구조를 활용해 프로그램의 행동을 설명하더라도 완벽한 프로그램을 만들기란 어렵다. 그럼에도 우리는 완벽에 가까워지는, 적어도 큰 문제는 발생하지 않을 프로그램을 만들기 위해 정적 분석 도구를 활용할 수 있다. 페이스북은 당사 정적 분석 도구인 Infer와 Zoncolan을 이용해 중요한 버그를 빠르고 효율적으로 찾아내 고칠 수 있었다. 해당 사례를 참조하여 이데아에 더 가까워질 수 있도록 개발자들을 돕는 도구는 어떤 방향으로 나아가야 할지 예상해 보았다. 코드의 수정을 반영시키는 시점에 재빠르게 분석 결과를 내주는 것에서 더 나아가, 개발자들이 코드를 짜는 과정에서 생산성을 저해시키지 않고 완벽에 더 가까운 프로그램을 만들어 내는 방법이 있을 것이라 생각한다.

우리가 관찰할 수 있는 모든 것은, 완벽한 존재인 이데아에 가까워질 수 있으나 이데아 그 자체일 수 없다. 플라톤이 주장했던 이데아론의 한 부분이다. 전산학을 어느 정도 공부하고 무언가 목적을 가지는 프로그램을 프로그래밍 언어로 구현해본 적 있는 사람이라면, 프로그래밍이야말로 이데아론의 비유로 적절한 활동이라는 생각에 동의하지 않을까 한다. 사람들은 어떤 문제를 해결하기 위해, 주어진 상황에 따라 알맞은 행동을 취하는 자동화 기계를 만들기 위해 프로그램을 구현한다. 그러나 사람이 구현하는 모든 프로그램은 의도하지 않은 행동, 더 나아가 버그와 취약점을 가지게 된다. 관념 속에서 완벽했던 존재가, 세상에 실재하는 순간 문제투성이가 되는 것이다.

우리는 프로그램의 이데아를 표현해내기 위해 프로그램이 해야 하는 행동에 대한 설명(specification)을 문서화한다. 가장 관념적인 설명 방법은 논리 구조를 활용하는 것이다. 그러나 사람이 논리 구조만으로 완벽한 설명을 하기란 프로그램을 잘 짜는 것 이상으로 어려운 일이다. 이론적으로, 완벽하게 기술된 논리 구조를 토대로 프로그램을 구현한다면 그 프로그램은 논리적으로 무결함을 증명할 수 있지만 논리 구조를 완벽하게 설명하는 데 소모되는 비용은 그야말로 배보다 배꼽이 커지는 수준일 것이다. 그렇기에 대부분의 경우 프로그램의 설명서는 자연어 또는 기대하는 입출력 예제를 통해 불완전하게 기술된다.

페이스북(Facebook)을 포함해 수많은 테크 기업과 개발자들은 이 불완전한 설명서와 결함있는 프로그램을 그래도 사용하는 데 큰 문제는 없게끔, 이데아에 최대한 가까울 수 있도록 하기 위해 정적 분석(static analysis) 도구를 활용한다. 정적 분석 도구는 이상적인 프로그램을 만들어주진 못해도 특히 중요하다고 판단한 기능(feature)과 버그(bug)에 대해 프로그램이 올바른 기능을 하고 버그를 일으키지 않는지 분석해 준다. 페이스북의 정적 분석 도구 Infer와 Zoncolan은 이 과정을 훌륭히 해내고 있다. 이 둘은 각각 페이스북의 모바일 앱, 서버에서의 코드를 분석하여 치명적인 수준의 버그를 자동으로 찾아내 주고 있다.

Infer와 Zoncolan이 성공적으로 적용될 수 있었던 이유 중 가장 인상깊었던 것은 코드 수정을 반영하는 시점에 재빠르게 분석 결과를 보고해주는, 변경 시점 분석(diff-time analysis) 방식이었다. 이들은 천만에서 억단위의 라인수를 가지는 코드를 빠르게 분석하기 위해 코드가 수정됨에 따라 의미가 달라지게 되는 부분만 추가 분석 과정을 거친다. 이 과정은 수행 시간 자체를 한 번에 15분 내외로 줄여줄 뿐만 아니라, 개발자들로 하여금 수정 이후 코드 리뷰(code review) 과정에서 문제를 파악할 수 있게 하여 개발자의 생산성에 악영향을 주지 않는다. 협업을 통해 프로그램을 개발해 나가는 과정에 정적 분석 도구를 자연스레 녹여낼 수 있었던 비결이지 않을까 한다.

다만, Infer나 Zoncolan과 같은 정적 분석 도구가 정말 모두를 만족시킬 수 있다고 하기에는 아직 해결해야 할 과제가 많이 남아 있다. 변경 시점 분석이 새로운 이슈를 보고하는 시점을 조정함으로써 분석 시간 단축과 개발자들의 생산성 유지를 모두 해냈다 하였지만, 그럼에도 잘못된 보고(false alarm 혹은 false positive)와 놓치는 버그(missed bug 혹은 false negative)의 존재는 개발자들에게 치명적이다. 디버깅 과정에 드는 인력 및 비용을 줄이기 위한 도구가 오히려 개발자를 피로하게 하는 요인이 된다면 그 도구는 있으나 마나 한 존재가

될 것이다. 하지만 잘못된 보고와 놓치는 버그를 모두 없애는 것은 불가능하다는 것이 입증되었기에, 우리는 분석기의 정확도를 높이는 것 이외에도 불완전(incomplete)하고 불안전(unsound)한 정적 분석 도구를 알맞게 활용할 방안을 찾아내야 할 것이다.

그 방안으로써 굉장히 빠른 미완성 코드 분석기를 제안하고자 한다. 만약 정적 분석 도구가 이미 완성된 프로그램, 또는 그러한 프로그램의 코드 수정을 반영하는 경우에만 분석 결과를 주는 것이 아니라 구현 중에 있는 미완성 프로그램의 코드 분석 결과를 빠르게 내줄 수 있다면 어떨까? 오늘날 대부분의 현대적인 언어에 탑재되어 있는 타입 체계(type system)를 활용한 타입 검사기(type checker) 역시도 정적으로 프로그램의 성질(property)을 파악해 위험하거나 불가능할 수 있는 구문(statement)과 식(expression)을 미연에 방지하는데, Infer와 같이 더 구체적인 결함을 파악하고 방지하는 기능 역시 타입 검사기처럼 초단위로 동작할 수 있다면, 개발자의 생산성에 큰 기여를 할 수 있을 것이라 믿는다. 반복문을 작성하는 도중 배열을 잘못 참조하여 발생 가능한 버퍼 오버런(buffer overrun)을 바로 찾아낼 수 있다면, 더 빠른 대처가 가능할 것이다. 또 이렇게 작성과 동시에 발생 가능한 결함들을 미연에 방지하기 때문에, 완성된 프로그램에 대한 분석 역시 상대적으로 효율적이고 안정적으로 가능할 것이고, 이는 결국 잘못된 보고와 놓치는 버그 역시 줄이는 효과를 볼 수 있을 것이라 기대한다.

이데아에 가까운 프로그램을 만드는 것은 개발자 혼자만의 힘으로는 한계가 있다. 페이스북의 Infer와 Zoncolan은 결함이 없는 프로그램을 만들어주진 못하지만, 개발자가 결함을 발견하고 고치는 데 큰 기여를 하며 그 과정에서 생산성을 떨어뜨리지 않도록 많은 도움을 주고 있다. 이 도구들이 자연스레 코드 수정 반영 단계에 스며든 것처럼, 각 개발자들의 에디터에 스며들어 안전한 프로그램을 빠르게 구현할 수 있는 도구의 개발 역시 가능하리라 믿는다.