코파일럿을 통해 바라본 차세대 프로그래밍

김태은

2022.03.11

Abstract

프로그램을 자동으로 생성하는 기술은 오래 전부터 컴퓨터 과학 분야의 성배로 여겨져왔다. 이 목표를 향한 연구자들의 열망은 지금까지 프로그램 합성과 패치 자동 생성 기술 등의 연구로 구체화 되어왔지만, 아직 실제로 사용할 수 있는 수준까지 발전하지는 못했다. 그러나 최근, 깃허브(GitHub)는 사용자의 코드를 자동으로 완성해주는 기술, 코파일럿(Copilot)을 발표하였다. 코파일럿은 마치 사람처럼 주어진 코드를 완성하지만, 마찬가지로 사람처럼 결함이 있는 코드를 작성하기도 한다. 이러한 문제에 대응하기 위해 기존에 사용되던 프로그램 오류를 검출하는 기법을 사용할수도 있지만, 이는 여전히 사람이 개발하고 사람이 디버깅하는 기존의 패러다임에 갖혀 있는 방식이다. 따라서 앞으로는 기계만이 할 수 있는 방법으로 프로그램을 생성하는 기술적 혁신이 이루어져야 할 것이다.

프로그램을 자동으로 생성하는 기술은 1940년대에 앨런 튜링[1]이 예견한 이후로 컴퓨터 과학 분야의 성배로 여겨져왔다. 프로그램을 자동으로 생성할 수 있다면 어떤 세상이 올까? 아마 이렇게 아이언맨에 나올 법한 대사가 자연스러운 일상이 될 것이다. "자비스, 새로운 설정으로 실험 돌려서 기존 결과 차트에 업데이트 해줘". 왜냐하면 이는, "새로운 설정으로 실험을 돌리고, 결과를 정리하는 프로그램을 작성해줘" 라는 말과 같은 말이기 때문이다. 따라서 수많은 사람들이 이를 고민하고 연구해 왔지만, 아직은 공상과학 영화에서나 볼 수 있는 기술이다.

앨런 튜링이 던진 지향점을 향한 연구자들의 열망은 지금까지 프로그램 합성과 패치 자동 생성 기술 등의 연구로 구체화 되어왔다. 프로그램 합성은 사용자의 의도가 입출력 예제나 논리식 등의 형태로 주어지면 이를 만족하는 프로그램을 합성해내는 기술이다. 대부분의 경우 주어지는 조건이 비교적 명확하고 이를 만족하는 것이 엄격하게 요구된다. 따라서 생성 가능한 모든 가능한 프로그램을 생성 순서대로, 혹은 확률적 분포를 고려해서 탐색하는 것이 일반적이다. 이는 많은 탐색 비용이 발생하기에 프로그램 합성 기술은 주로 최적으로 설계된 작은 언어와 함께 사용된다. 그에 비해 보다 간접적으로 앨런 튜링의 비전을 쫓는 자동 패치 생성은, 주어진 결함을 수정하는 코드를 자동으로 생성하는 기술이다. 전체 코드의 일부만 수정하면 되기에 쉬워 보이지만, 이 또한 무수한 가능성을 탐색해야 하기에 본질적으로 그 난이도는 프로그램 합성과 크게 다르지 않다. 따라서 두 기술 모두 아직 일반적인 프로그램 개발에서 사용하기에는 현실적인 한계가 존재한다.

그러나 최근, 깃허브(GitHub)는 사용자의 코드를 자동으로 완성해주는 기술, 코파일럿(Copilot)을 발표하였다. 깃허브는 가장 크며 널리 상용되는 소스코드 저장소 서비스를 제공하는 회사로서, 코파일럿은 깃허브 저장소의 수많은 코드를 학습하여 만들어진 인공지능 도구이다. 코파일럿은 이렇게 수많은 개발자들의 코드를 학습한 덕분에 사용자가 코드를 일부만 작성하여도 그 의도와 함께 이후에 올 코드를 예측하여 프로그램을 완성할 수 있는 것이다. 이 기술은 프로그램 합성이나 자동 패치 생성과는 달리 실제로 개발자들의 능률 향상에 큰 도움을 줄 수 있다. 엄밀한 조건과 테스트를 통과하지 않아도, 다른 개발자들의 코드로부터 얼추 맞는 코드를 생성하는 방식이 그 열쇠였던 것이다. 하지만 코파일럿이 학습한 코드가 모두 훌륭한 스승은 아니었다.

코파일럿은 사람의 의도를 예측하고 사람처럼 코드를 완성하지만, 역시 사람처럼 결함이 있는 코드를 작성하기도 한다. 이는 코파일럿의 태생적인 한계로서, 깃허브 저장소에서 추출한 수많은 학습데이터에 오류가 있는 코드도 많지만 이를 사람이 모두 검수할 수 없었기 때문에 발생한 문제이다. 이렇게 인공지능에 의해 등장한 버그는 더 은밀하게 우리 프로그램을 위협한다. 우선 사용자 입장에서 매번 사용하는 인공지능 도구의 추천 코드를 의심하는 것 자체가 쉽지 않다. 또한 자동 완성된 코드는 사용자가 직접 작성한 코드가 아니기에, 이를 제대로 이해하고 결함을 찾아내는 것도 어려운 일이다. 더 나아가 누군가는 의도적으로 악성 코드를 다음 버전의 코파일럿에게 학습시키려 할 수도 있다.

[1] Turing, A. M., Proposed electronic calculator, 1946

이 때, 프로그램 오류를 검출하는 기법을 자동화하여 사용한다면 코파일럿의 이런 태생적 한계를 보완할 수 있을 것이다. 코파일럿이 등장하였어도 지금까지의 프로그램 개발과 디버깅의 패러다임이 근본적으로 변하지 않았기 때문에 우리는 기존의 프로그램 오류 검출 기법을 그대로 사용할 수 있다. 왜냐하면 여전히 사람의 코드를 학습하여 그대로 코드를 생성하는 방식이기 때문이다. 다만 자동화가 필요한 이유는 개발자가 작성할 수 있는 코드의 양이 이전보다 많아짐으로써 기존에 발생하던 프로그램의 결함 문제들이 양적으로 더 많아질 것이기 때문이다. 이러한 프로그램 결함의 양적 증가에 대응할 기술에는 프로그램 정적분석과 퍼징등이 있다. 프로그램 정적분석은 프로그램을 실행하지 않고 분석하여 취약점으로 의심되는 부분을 찾아내는 기술이다. 또한 퍼징은 자동으로 생성한 수많은 입력으로 프로그램을 실행하여 결함을 찾아내는 기술인데, 그 중 지향성 퍼징은 특정한 목표 지점에 더 집중할 수 있는 특징이 있다. 이 두 기술을 결합한다면, 정적분석을 통해 프로그램의 결함이 의심되는 지점을 찾아내고, 지향성 퍼징으로 그 지점을 집중적으로 검사할 수 있을 것이다.

그러나 이러한 기술들로 코파일럿을 보조하는 방식은 문제의 근본적인 해결책이 될 수 없다. 따라서 앞으로는 기존의 방법을 완전히 벗어나서 새로운 방식으로 프로그램을 생성하는 기술이 필요할 것이다. 일단 자동으로 오류가 있는 프로그램을 생성한 이후에, 자동으로 그 프로그램을 검사하거나 수정하는 방식은 기존에 사람이 하던 일을 그대로 가져와서 기계가 대신 하는 수준에 머물 뿐이다. 마치 고속도로 요금 징수의 자동화를 위해 사람의 역할을 그대로 대체하는 로봇팔을 배치하는 것과 같은 것이다. 반면, 하이패스는 사람이 하던 일을 그대로 가져오는 패러다임에서 벗어난 기술이었기에 혁신적인 기술이 될 수 있었다. 프로그램 개발에 있어서도 이런 혁신적인 접근의 기술이 필요하다. 따라서 우리 연구자들은 현재 주어진 기술인 코파일럿을 안전하게 사용하는 방법을 고안하되, 그와 동시에 여기에 종속되지 않고 새로운 돌파구를 찾는 노력을 해야 할 것이다.