AI와 프로그램 논증

이서현

Abstract

이 글은 AI의 한계와 프로그램 논증 기술와 결합한다면 어떠한 결과가 나올지에 대해 다룬다.

프로그래머들은 언제나 더 좋은 언어를 만들기 위해 노력해왔다. 인간은 다른 동물과 차별화시키는 것은 바로 도구를 사용하고, 의사소통한다는 것이다. 물론 다른 요소도 중요하지만, 인간을 현재의 위치로 데려다 놓은 가장 중요한 요소들은 이 둘이라 생각한다. 인간은 도구를 발전시키면서 진화해왔고, 프로그래머도 인간이다. 프로그래머도 자신의 프로그램을 만들기 위한 도구인 프로그래밍 언어를 점점 발전시키고 있다.

진공관에 전선을 연결하는 것으로부터 시작됐던 프로그래밍은 천공카드를 거쳐 마침내 폰 노이만 구조, 즉 프로그램을 동적으로 수정할 수 있게 되었다. 그 때부터 프로그래밍 언어가 생겼다. 단순히 기계어를 사람이 읽을 수 있도록 옮긴 어셈블리부터 시작해 코볼, C와 같이 점점 발전한 프로그래밍 언어는 나중에는 Matlab와 같이 특수한 목적을 위해 사용되는 언어도 발생하기 시작했다. 더욱 좋은 프로그래밍 언어를 만들기 위해, 프로그래머들은 노력한다. 모질라 팀은 C++의 포인터 불안정성을 극복하기 위해 Rust라는 새로운 언어를 개발했다. 꼭 새로운 언어를 만드는 것이 좋은 프로그래밍 언어를 만드는 것은 아니다. 구글은 V8 엔진을 만들어 자바스크립트를 단순한 웹 효과 부여용 스크립트에서 파이썬 만큼의 유연함에 컴파일 언어 만큼의 속도를 갖도록 진화시켰다. 이처럼, 좋은 언어를 위한 프로그래머들의 노력은 끊기지 않는다.

하지만, 중요한 것이 있다. 어떤 것이 좋은 프로그래밍 언어인가? 사람마다 좋아하는 프로그래밍 언어는 다르다. 필자와 같이 웹 개발을 주로 했던 사람들은 고성능이면서 유연한 자바스크립트와 같은 언어를 좋아할 수도 있고, 임베디드 시스템을 다루는 개발자에게는 어셈블리 외에 다른 선택지는 애초에 없을 것이다. 이처럼 개발자는 사용하는 목적에 따라 선호하는 프로그래밍 언어, 더 나아가서는 패러다임이 모두 다르다. 이것이 새로운 언어가 계속 개발

이러한 언어는 의사소통을 위한 도구이다. 만일 세상에 유일무이한 존재가 있다면 생각만으로 충분할 것이다. 하지만, 우리는 주위의 사람들과, 더 나아가서 우리가 만든 기계들과 함께 작업을 수행한다. 이러한 과정에서 발생한 것이 언어이다. 인간의 언어, 컴퓨터의 언어는 모두 시간이 지남에 따라 발전해 왔다. 그러나, 발전의 방향성은 다소 차이가 있다. 그 이유는 당연하게도 인간과 컴퓨터의 차이 때문이다. 인간의 언어는 결국 인간이 쓰고 말하기 쉽게 만들어졌다. 이로 인해 다양한 예외가 추가되었다. 반면, 컴퓨터의 언어는 매우 엄격한 타입을 가진다. 그 이유는 컴퓨터가 인간보다 멍청하기 때문이다.

컴퓨터는, 적어도 현재 상용적으로 사용되는 수준에서는 우리가 시킨 일밖에 하지 못하는 멍청이이다. 이것은 현재의 한계이다. 현재로써는 정확성과 유연성 모두를 챙길 방법이 없다. 당장 인간조차 이러한 문제를 겪는다. 인터넷 커뮤니티에 주기적으로 올라오는 문제가 있다.

"48/2(9+3)은 무엇인가?"

어떤 사람들은 이 수식의 값이 2라고 하고, 다른 사람들은 288이라고 한다. 그리고, 실제 정답은 "정의되지 않음"이다. 일반적으로 괄호의 결합 순서는 정의되지 않았고, 이를 먼저 결합할 지 아니면 나눗셈을 먼저 결합할 지는 수학에서 정의되지 않았기 때문에 애초에 수식으로서 성립하지 않는 것이다. 즉 프로그래밍 언어에 대입한다면 일종의 문법 오류를 발생시킬 것이다. 하지만, 이것이 중요한 시험이 아니라 SNS와 같은 곳에 올라왔다면 아무 문제도 발견하지 못하고 "2"라는 답을 낼 수도 있을 것이다. 요컨데, 언어의 편의성을 높이게 된다면 해석에 관해 충돌이 일어날 것이고, 정밀성을 높이게 된다면 매우 복잡하게 될 것이라는 말이다.

물론 ChatGPT와 같은 AI의 발전으로 사용자의 문맥을 읽어 우리가 지시하지 않은 사항까지 자연스럽게 처리하려는 인공지능에 대한 개발은 계속되고 있으나, 아직은 불완전하다. 간단한 프로그램을 짠다고 하더라도 대부분의 경우에는 맞을 것이나 이것이 항상 옳다는 보장이 없다. 이것이 현재 AI 기반 프로그래밍의 가장 큰 문제이다. 물론, 현재의 수준 정도의 AI만 되더라도 프로그래밍에 매우 큰 도움이 될 수는 있다. 특히 UI 작업/DB ORM 작업과 같은 단순 반복 작업에서는 한 번도 안 써본 사람은 있어도 한 번만 써본 사람은 없다는 말이 나올 정도로 이미 매우 유용하게 사용되고 있다. 그러나, 이러한 사용 영역은 다시금 AI의 한계를 증명하는 꼴이다. AI가 생성한 코드는 믿을 수 없다. UI와 같이 바로 눈으로 보이는 요소나 DB 요청과 같이 강력한 타입을 가져 린트

한번 검증이 끝나는, 다시 말해 우리가 사실인지 바로 알 수 있는 코드에 대해서만 주로 쓰이고 있다는 것이다. 모든 곳에 AI를 도입했다가는 10000원을 입금했는데 잔액에서 10000원이 사라지는 일들이 발생할 것이다.

하지만, 달리 말한다면 AI의 결과물에 대한 안정성만 보증된다면, 프로그래밍 언어와 AI의 통합은 놀라운 발전을 가져올 것이다. AI가 인간의 언어를 이해하고, 그에 맞춰 코드를 생성하거나 수정하는 능력이 향상되면, 프로그래머의 업무는 혁신적으로 변화할 것이다. 예를 들어, 자연어로 명령을 내리더라도 위에서 언급한 정밀성을 AI가 알아서 판단해 코드를 작성하게 된다는 것이다.

내 짧은 식견으로는, 지금 배우고 있는 프로그램 논증이 이것에 대해서 다루는, 또는 다루게 될 분야라고 생각한다. AI가 작성한 코드에 대해, 이것이 정말로 문제가 없는 코드인지 검증을 진행하는 것이다. 이렇게 된다면 리스크 없이 AI의 가능성을 모두 사용할 수 있을 것이다. 실제로, 이러한 노력이 이미 이뤄지고 있다고 알고 있다. 예를 들어, 깃허브에서 개발한 CodeQL이 있다. 이전에 한번 사용해 본적이 있어서 이름은 알고 있었는데, 이것이 왜 의미가 있는지 이번에 조사를 하면서 알게 되었다. 만일 AI에 대해 안정성이 보장된다면, AI를 이용하여 코드를 생성하는 것에서 멈추는 것이 아니라, 아예 AI 기반의 프로그래밍 언어를 만드는 것도 가능할 것이다. 의사 코드와 같이 논리와 원하는 기능을 섞어 쓰면, AI가 자동으로 이를 안정적으로 구현할 수 있지 않을까 한번 조심스럽게 생각해 본다.

프로그래밍 언어와 AI 기술의 융합은 미래의 프로그래밍 패러다임에 혁신적인 변화를 가져올 것이다. AI의 발전은 프로그래머들에게 새로운 가능성을 열어주고 있으며, 이는 단순히 코드 생성을 넘어서 프로그래밍 언어 자체를 변화시킬 수 있다. 예를 들어, 의사 코드와 같이 논리와 기능을 혼합하여 작성하면, AI가 이를 안정적으로 구현할 수 있는 새로운 프로그래밍 언어의 등장도 상상해볼 수 있다.

이런 미래에서는 프로그래머들이 더 이상 복잡한 문법이나 구조에 신경 쓰지 않고, 원하는 기능과 논리를 명확히 표현하기만 하면 된다. AI가 이러한 요구 사항을 이해하고, 최적화된 코드로 변환하는 것이다. 이 과정에서 AI의 안정성과 신뢰성은 필수적이다. 이를 위해 AI가 작성한 코드에 대한 검증과 테스팅이 중요한 역할을 할 것이다. 이미 CodeQL과 같은 도구들이 이러한 방향으로 진행되고 있는 것은 이 분야의 발전 가능성을 더욱 확신시킨다.

결국, AI와 프로그래밍 언어의 결합은 단순한 자동화를 넘어 프로그래밍의 본질을 변화시킬 것이다. 이는 프로그래머들이 더 창의적이고 효율적으로 작업을 할 수 있는 환경을 만들어 줄 것이며, 인간과 기계의 상호작용을 더욱 자연스럽고 직관적으로 만들 것이다. 이것이 바로 프로그래밍 언어와 AI의 통합이 우리에게 가져다 줄 미래의 모습이다.