불완전하지만 유용한

이동재

소프트웨어의 불완전함은 괴델의 불완전성 정리와 튜링의 멈춤 문제를 통해 이미 증명되었다. 그러나 소프트웨어는 현대 사회의 근간을 이루며, 우리의 삶에 필수적인 유용성을 제공하고 있다. 소프트웨어 공학자들은 이러한 불완전함을 최대한 억제하고 유용성을 극대화하기 위해 정적 분석, 동적 분석, 검증, 테스트 등 다양한 기술을 개발해왔다. 근본적으로 불완전한 시스템 속에서 유용한 소프트웨어를 창조하는 것은 소프트웨어 공학의 아름다움이자 커다란 도전이다. 본 글에서는 소프트웨어의 불완전함을 극복하기 위한 미래의 방향성을 예측하고, 그 과정에서 소프트웨어 공학 전문가로서 우리가 수행해야 할 역할을 논의하고자 한다.

소프트웨어의 근본적인 한계는 괴델의 불완전성 정리와 튜링의 멈춤 문제를 통해 명확히 드러난다. 괴델은 일관적이고 충분히 복잡한 형식 체계 내에서는 참이지만 증명할 수 없는 명제가 존재함을 증명하였다. 이는 수학적 시스템의 완전성이 불가능함을 의미하며, 컴퓨터 프로그램 역시 이러한 논리 체계 위에서 동작하기 때문에 모든 프로그램의 올바름을 완벽히 증명하는 것은 불가능하다. 튜링의 멈춤 문제는 어떤 일반적인 알고리즘으로도 모든 프로그램이 멈출지 아닐지를 결정하는 것이 불가능하다는 것을 보여준다. 이는 프로그램의 예측 불가능성을 내포하며, 소프트웨어의 완전한 검증을 어렵게 만드는 핵심 요인이다.

그럼에도 불구하고, 소프트웨어는 현대 사회에서 필수불가결한 역할을 수행하고 있다. 소프트웨어는 우리의 삶을 편리하게 해주는 도구이며, 스마트폰, 컴퓨터, 가전제품 등 거의 모든 전자기기에 탑재되어 있다. 전기를 사용하는 거의 모든 물건에는 이를 제어하는 소프트웨어가 내장되어 있다고 해도 과언이 아니다. 소프트웨어의 불완전함은 그 유용성을 제한하지만, 그것이 소프트웨어가 불필요하다는 뜻은 아니다. 오히려, 이러한 불완전함을 극복하고 소프트웨어의 유용성을 극대화하기 위한 노력은 소프트웨어 공학의 발전을 촉진시켰다.

소프트웨어의 불완전함을 극복하기 위해 소프트웨어 공학자들은 정적 분석, 동적 분석, 검증, 테스트 등 다양한 기술을 개발해왔다. 정적 분석은 프로그램 코드를 분석하여 잠재적인 오류를 발견하는 방법으로, 모든 경우를 고려할 수 있지만 항상 거짓 경보(false positive)를 발생시킬 수 있다. 예를 들어, 변수 사용 전에 초기화되지 않았는지 확인하거나, 배열의 경계를 넘어서 접근하는 코드가 없는지 검사한다. 동적 분석은 프로그램을 실제로 실행하여 오류를 찾아내는 방법으로, 거짓 경보는 없지만 모든 상황을 고려하지 못한다는 한계가 있다. 예를 들어, 특정 입력에 대해서만 발생하는 버그는 발견할 수 있지만, 테스트하지 않은 입력에 대해서는 알 수 없다. 이러한 기술들은 상호 보완적인 특성을 지니며, 이를 적재적소에 활용하여 소프트웨어의 품질과 신뢰성을 향상시키고 있다.

하지만 현재의 소프트웨어 공학 기술로도 프로그램의 안전성을 완벽히 보장하기는 여전히 어렵다. 현대의 소프트웨어 시스템이 지나치게 복잡하고 규모가 크기 때문이다. 예를 들어, 항공기의 비행 제어 소프트웨어나 원자력 발전소의 제어 시스템은 수백만 줄의 코드로 이루어져 있으며, 이들 시스템에서 발생하는 작은 오류 하나가 치명적인 결과를 초래할 수 있다. 소프트웨어의 복잡도와 규모는 갈수록 증가하고 있으며 그 활용 범위 역시 넓어지고 있다. 따라서, 다음 세대의 소프트웨어 공학은 새로운 기술과 방법론이 필요할 것이다.

미래에는 확률 기반의 시스템이 소프트웨어 공학의 핵심이 될 것으로 예상된다. 확률 기반 시스템은 불확실성을 다루는 데 강점을 지니며, 소프트웨어의 불완전함을 극복하는 데 큰 도움이 될 것이다. 예를 들어, 머신러닝 알고리즘은 대량의 데이터를 학습하여 예측 모델을 생성하며, 이는 완벽하지는 않지만 높은 확률로 정확한 결과를 제공한다. 소프트웨어의 불완전함은 특정 성질을 확실하게 보장할 수 없다는 것을 의미하지만, 그 불확실성이 완전한 무질서를 의미하는 것은 아니다. 특정 방향으로 편향된 무작위성을 다루는 데 최적화된 것이 확률 기반 시스템이며, 소프트웨어에 최적화된 확률 시스템을 고안하는 것이 핵심 과제가 될 것이다.

현재로서는 기존 방법론에 확률을 첨가하거나, 자연어를 통해 거대한 언어 모델속 지식을 간접적으로 활용하는 것이 일반적이다. 그러나 확률이 소프트웨어와 완벽히 융합되기 위해서는

근본적인 변화가 필요하다. 단순히 자연어를 통해 언어 모델에 명령을 내리는 형태가 아니라, 추상 구문 트리(Abstract Syntax Tree), 실행 흐름 그래프(Control Flow Graph), 정의-사용 그래프(Def-Use Graph) 등의 소프트웨어 구조 정보를 입력으로 받을 수 있는 다중 입력(Multi-Modal) 모델을 구축하는 것이 그러한 노력의 한 예가 될 수 있다. 한 발 더 나아가면, 소프트웨어에 최적화된 거대 확률 모델을 구축하는 것이 가능해질 것이다. 이러한 모델은 소프트웨어의 불완전함을 극복하고, 소프트웨어의 유용성을 극대화하는 데 큰 역할을 할 것이다.

소프트웨어가 탄생한 이래로, 소프트웨어 공학자들은 그 불완전함을 극복하기 위해 부단히 노력해왔다. 그러나 그에 발맞추어 소프트웨어의 크기와 복잡도는 점점 거대해지고 있으며, 우리는 불완전함의 늪에 점점 빠져들고 있다. 결국 기존에 유용했던 해결책도 시간이 지나면 더 이상 유효하지 않게 된다. 따라서, 소프트웨어 공학자로서 우리는 끊임없이 변화하는 소프트웨어 세계의 불완전함을 인정하고, 이를 극복하기 위한 혁신적인 해결책을 창조해내야 한다. 이러한 노력은 소프트웨어의 유용성을 극대화할 뿐만 아니라, 더 안전하고 신뢰성 있는 시스템을 구축하는 데 필수적이다. 앞으로도 다양한 시도가 이루어질 것이며, 우리는 그 최전선에서 소프트웨어의 미래를 이끌어 나가야 할 것이다.