컴퓨터란 무엇인가?

2011160004

이준희

나는 항상 컴퓨터가 무엇인지 그 대답에 대한 갈증이 있었다. 복수전공을 하게 되었지만, 컴퓨터가 무엇인지 몰라 내가 여기서 과연 무엇을 할 수 있고 무엇을 해야 할 지 감이 잡히지 않았다. 이 책은 컴퓨터가 무엇인가, 그에 대한 대답을 주기 위해 쓰여졌다. 이 책은 그렇게 컴퓨터가 어떠한 발상에서 디자인 되었고, 그것을 단계별로 구현하여 컴퓨터를 만들어내고 그 컴퓨터가 만들어내는 세상에 대한 이야기다. 나는 이번 기회로 컴퓨터가 무엇인지, 그 한계가 무엇인지를 이해하고 컴퓨터로 문제를 해결한다는 게 무엇인지 그 의미를 찾아가려고 한다.

컴퓨터가 무엇인가에 대한 대답을 가장 잘 주는 것은 아마도 컴퓨터의 원형인 Universal Turing Machine 이라고 생각한다. 튜링은 기계로 모든 문제를 풀 수 없다는 것을 보이기 위해 어떠한 기계를 디자인 하였다. 그리고 그 기계가 모든 기계가 할 수 있는 일을 할 수 있는 궁극적인 기계임을 보이는데 그것이 Universal Turing Machine 이다. 어떻게 그런 일이 가능하는가 하나면, UTM 은 임의의 기계 Turing machine 을 입력 받아 그것을 해석하여 그 기계의 규칙대로 실행할 수 있는 디자인을 갖추었기 때문이다. 그리고 이것이 바로 우리가 사용하는 컴퓨터이다. 우리가 임의의 소프트웨어 TM 을 만들면 UTM 인 컴퓨터는 주어진 TM 을 받아 해석하여 그대로 실행시켜준다. 즉, 컴퓨터는 주어진 TM 을 해석하여 실행시켜주는 Interpreter 라고 볼 수 있다. 이렇게 임의의 기계를 받아들이기 때문에 컴퓨터는 때로는 계산기가 될 수도, 주어진 영상을 재생시켜 화면에 보여주는 영상기가 될 수도 있다. 또 이러한 관점에서 우리가 들고 다니는 스마트폰도 App 을 해석하여 실행시켜주는 Interpreter, 즉 컴퓨터라고 볼 수 있겠다.

컴퓨터를 이러한 관점에서 보면, 컴퓨터를 이용하여 문제를 푼다는 것은 TM 을 만든다는 것이 된다. 그렇지만 우리는 TM 을 직접 만들지 않고 주어진 프로그래밍 언어에 알고리즘을 짜 넣는 일을 한다. 이러한 일이 가능한 것은 프로그래밍 언어라는 번역 사슬이 있기 때문이다. 문제를 푸는 방법을 주어진 언어로 기술하면, 컴파일러들이 그 언어를 차례 차례로 기계어 = TM 으로 번역해준다. 즉, 컴퓨터로 문제를 해결한다는 것이 단순히 타이핑을 통해 알고리즘을 언어에 맞게 기술한다는 것이 된다. 컴퓨터가 발명되기 이전에는 특정 문제를 해결한다는 것은 손으로 직접 계산을 하거나 노동, 기계를 만든다는 것이었다. 그것이 컴퓨터의 발명에 의해 알고리즘을 짜고 그것을 코딩하는 과정으로 바뀐 것이다. 그러나 그것이 앉아서 모든 문제를 해결할 수 있다는 것을 의미하진 않는다. 해결할 수 없는 문제들이 많기 때문이다.

컴퓨터 = UTM 은 임의의 TM 을 받아 실행시킬 수 있지만, 모든 문제를 해결할 수 없다. 애초에 UTM 은 계산 불가능성을 증명하기 위해 만들어진 도구다. 그렇다면 모든 문제를 해결할 수 없다

는 게 무슨 뜻일까? TM 의 정의에 따라 해석하면 임의로 주어진 스트링의 집합을 accept 하는 TM 이 존재하지 않을 수 있다는 말이 된다. 그리고 이러한 문제들이 풀리는 문제보다 셀 수 없을 만큼 많다는 것이 증명되어 있다. 이 말은 임의로 주어진 스트링의 집합, 내 입맛대로 맞는 결과물을 도출하는 TM 은 대부분 존재하지 않는다는 뜻이 된다. 게다가 가상의 TM 으로 해결이 가능하더라도 그것을 Digital Computer 로 해결하려고 하면 현실적인 비용으로 해결이 되지 않는 문제들도 많다. 이에 따라 실제로 해결할 수 있는 문제의 범위는 더욱 좁아지고, 어쩌면 컴퓨터로 완벽하게 풀 수 있는 문제란 생각보다 정말 적을 수도 있다.

그렇다면 TM 을 확장하는 방법은 없을까? TM 을 수학적으로 보면 {0,1}* 의 부분집합 아래 정의된 함수이다. 그리고 TM 들의 집합이란 이러한 함수들의 집합이다. 이 TM 들은 서로 순서쌍으로 묶거나 하는 연산을 해도 여전히 TM 이고, TM 은 0 과 1 의 string 으로 변환이 되는 것을 알고 있다. 즉, TM 들의 집합은 어떠한 {0,1} 로 이루어져 있고, 거기서 어떤 연산을 하여도 다시 TM 이 된다는 것이다. 즉, 확장이 안된다는 것이다. 이러한 TM 들의 집합은 유리수들의 집합과도 닮았다고 볼 수 있다. 유리수들은 + 와 * 가 잘 정의되는 이성적인, rational number 이지만 incomplete 하다. 즉, 빠진 수들이 있다. TM 들도 그러하다. 유리수에서는 이러한 상황에서 극한을 정의함으로써 실수들의 집합으로 넘어갔다. 그렇다면 TM 들도 극한을 생각해볼 수 있을까? 한가지 무한대의 스트링을 인풋으로 받는 TM 을 생각해볼 수도 있는데, final state 를 무한히 지나가면 accept 하는 걸로 정의하는 것이다. 그렇지만 이러한 것들은 현실적으로 불가능하니 번뜩이는 아이디어 없이는 당장은 실현 불가능할 것이다.

그러나 한가지 분명한 것은 컴퓨터는 계산은 정말 잘한다는 것이다. 무언가를 세는 능력, 비교하는 능력, 숫자를 계산하는 능력 등은 사람들보다 월등히 잘하고, 실수 또한 없다. 이러한 간단한 계산 능력들이 수학을 만나게 되면 어마어마한 일들을 할 수 있게 된다. 수많은 웹 페이지들 속에서 내가 원하는 페이지를 찾는 문제가 그렇다. "좋은 페이지" 가 사람들이 많이 방문하는 페이지로 일단 정의하고 나면, 그것을 수학적 영역으로 끌고 올 수 있게 된다. 그 이후는 Marcov Chain 모델을 이용하면 컴퓨터는 주어진 계산만 하면 된다. 그러면 페이지들마다 랭크를 매길 수 있게 되고 그 순서대로 결과물을 보여주면 된다. 그것이 구글이다. 이렇게 보면 컴퓨터를 이용하여 문제를 해결한다는 것이 또 달라진다. 단순히 주어진 언어에 맞게 코딩하는 것이 아닌, 문제를 잘 정의하고 이를 수학적으로 접근하여 알고리즘을 만드는 것이 컴퓨터로 문제를 해결하는 방법이 된다.

컴퓨터는 분명 대단한 계산 기계임에는 분명하지만, 한계 또한 갖고 있다. 그 한계를 명확히 인지해야 한다. 할 수 있는 것과 할 수 없는 일을 구분 짓고 수학적 아이디어를 바탕으로 알고리즘을 개발하는 일이 컴퓨터로 문제를 해결하는 방법일 것이다. 또한, 이러한 알고리즘이 쉽게 짜일 수 있도록 프로그래밍 언어를 개발하는 것이 컴퓨터가 가진 한계를 극복하는 방법이라고 생각한다.