또 하나의 '세계'를 만들기까지

우영호

책 <컴퓨터과학이 여는 세계>에서 흥미로운 것은 항상 컴퓨터와 자연의 '세계'를 비교하고 있다는 점이다. 기존의 도구들이 자연에서 파생된 하나의 도구에 불과했다면, 지금 계속해서 설명하고 있는 컴퓨터는 '자연'과 유사한 혹은 동등한 층위에서 또 하나의 세계를 창조하고 있다는 점을 계속 견지하고 있는 것이다. 따라서 컴퓨터의 세계와 자연의 세계, 그 특질들을 공통점과 차이점을 밝혀내고 이를 비교하고 있는 것이 흥미롭다.

이런 관계에 대해 책에서는 여러 가지를 제시하고 있는데 그 중에서도 먼저 부울 식을 소개하고 있다. 부울 식은 사람의 생각을 3가지 연산자 즉 and, or, not으로 모두 구현(조립)할 수 있다는 논리이다. 이런 논리가 스위치로 구현되고 이것이 데이터를 처리하는 기본 구조가 된다.

재밌는 것은 이런 논리회로가 차곡차곡 쌓기(hierarchy)에 의해 구성되고 있다는 점이다. 이 단계에서는 바로 아래 단계의 물건들을 사용해서 보다 큰 물건을 만들고, 그다음 윗단계에서는 이렇게 만든 물건으로 그 윗단계의 큰 물건을 만든다. 이런 일들이 각 단계에서 반복되면서 최종적인 목표물이 제일 윗단계에서 만들어진다. 그런데 이 차곡차곡 쌓기가 속내용을 감추면서(abstraction) 이루어지기 때문에 효율적이다. 이 속마음을 감추며 차곡차곡 쌓기(abstraction hierarchy)는 컴퓨터 부품, 디지털 논리회로뿐만 아니라 계속해서 전개되는 컴퓨터의 동작 원리에 광범위하게 적용된다는 점에서 흥미롭다고 할 수 있다. 사실 이 원리는 현실에서도 많이 일어나지 않는가. 현실세계의 원리와 컴퓨터 세계의 원리가 여기서 또한 대칭된다.

이런 토대 위에서 우리가 소프트웨어를 다루는 법은 무엇이 있을까? 책에서는 크게 알고리즘과 언어라는 두 가지 방법을 잡고 있다. 만들려는 소프트웨어가 일하는 방도와 그 소프트웨어를 표현하는 방식이다.

알고리즘은 우리가 어떤 문제를 컴퓨터로, 기계적인 방식으로 자동으로 돌릴 수 있는 풀이법을 말한다. 이 책에서는 알고리즘에 관해 몇 가지 문제를 제시한다. 문제가 컴퓨터로 풀 수 있는 것들인가?(멈춤문제 등) 혹은 풀 수 있어도 그 비용이 너무 많이 들지는 않는가? 즉 컴퓨터에서 어떤 문제를 처리할 때 소모되는 시간이나 메모리라는 비용, 즉 복잡도를 효율화하기 위해서는 어떻게 해야 할지가 관건이다.

비용과 관련해 문제를 제시하는 것은 간단하게 현실적인 알고리즘과 비현실적인 알고리즘으로 나눌 수 있겠다. 비용이 견딜만한 현실적인 다항 알고리즘(P클래스 문제)과 입력 크기가 커질수록 비용이 기하급수적으로 증가해 견딜 수 없는 비현실적인 알고리즘으로 나눌 수 있는 것이다. 재밌는 것은 '운에 기대면 현실적인 비용으로 해결할 수 있는' NP클래스 문제가 있다는 것이다. '어려운' 것이 무엇인지는 정확하게 정의할 수는 없지만 우리가 생각하기에 아직 운에 기대지 않고는 비용이 너무나도 많이 드는 문제다. NP클래스를 이해하기가 쉽지는 않았는데 풀기는 '어려울' 수 있어도 답이 맞는지 검산하는 것은 쉬운 문제라고 생각하면 명료해진다. P가 NP와 같은지 아닌지에 대해 아직도 확실히 밝혀지지 않았다는 점(이것을 밝혀내기 위해 많은 수학자가 도전했지만 번번이 실패하였다고 한다)과 NP문제를 모두 관통할 수 있는 NP-완전문제가 있다는 사실이 흥미롭다.

이런 문제를 '정직하게' 푸는 것이 어려우니까, 우리는 다양한 '꼼수'들을 이용해 풀 수 있다.

요구조건을 좀 더 느슨하게 만들어 비용을 현실적으로 만드는 '통밥', 통밥의 낭패를 막아줄 '무작위' 알고리즘이 그런 것이다. 컴퓨터과학이라고 한다면 뭔가 항상 철저한 논리구조 속에서 정확한 답을 요구하는 것 같지만, 의외로 컴퓨터 속의 세계에서도 '적당히'라는 인간적인 요소가 큰 도움이 될 수 있는 것이 재밌다.

소프트웨어를 담는 그릇인 언어도 이런 관점에서 바라볼 수 있다. 여기서 언어란 소프트웨어를 만드는 작업인 프로그래밍 언어에 대해 얘기하는 것이다. 사람이 알고리즘을 찾아 소프트웨어를 만들고, 컴퓨터는 그 소프트웨어를 실행하는 과정에서 언어가 필요하다. 사람이 쓰는 언어와 컴퓨터가 최종적으로 알아듣는 기계어는 다르니까(컴퓨터가 'goto'라는 말을 최종적으로 기계어로서 알아듣지는 않을 것이니까) 사람의 언어와 컴퓨터의 기계어, 그 간격을 메꿔주는 것이 필요한 것이다. 그리고 이 간격은 상위언어와 번역으로 메꿔진다. 자연어의 번역은 그 의미가 다의적으로 파생되는 경우가 많기 때문에 어렵지만, 프로그래밍 언어 사이의 번역은 의미가 '대응'이 가능하기 때문에 쉽다. 조립식과 불변성질 유지라는 두 가지 원리로 번역이 자동으로 이루어질 수 있다는 것이 큰 특징이다.

재밌는 것은 이런 프로그래밍 언어에 논리를 보태 더 다채로워질 수 있다는 것이다. 기계의 상태변화라는 것은 기계의 등장과 작동원리부터 그러했듯이 결국 데이터가 처리되는 것이 물리적으로 표현되는 것이다. 이는 역으로 기계와 전혀 상관 없어보이는 수학적 논리들이 프로그램의 세계를 더 효율적으로 또는 풍요롭게 만들 수 있다는 것이 된다. 책에서 람다 계산법은 이 말을 대변하고 있다. 람다 계산법은 기계적 계산을 식으로 표현할 수 있다. 람다 계산법의 관점은 결국 컴퓨터 세계에서 언어와 논리는 동전의 논리일 뿐 같은 것이라는 결론을 내게 된다. 즉 논리추론의 징검다리들이 컴퓨터 프로그램의 조립 방법과 대응한다는 것이다.

람다식은 튜링기계와 그 계산 능력이 정확하게 같아서 '기계적인 계산'을 수행할 수 있다. 그러면서도 다른 방식으로 프로그램의 실행을 상상하며 소프트웨어를 만들도록 유도한다. 튜링 기계에서부터 상위로 올라오는 언어들은 기계를 생각하는 프로그래밍 언어가 된다. 이 언어로는 기계의 메모리에 데이터를 읽고 쓰며 기계의 상태가 변해가는 상황을 머릿속에 그려가면서 프로그래밍하게 된다. 하지만 람다계산법에서 상위로 올라오는 언어들은 기계를 염두에 두지 않고 값들이 함수 사이를 통과하며 새 값으로 만들어지는 과정만 있다. 이 방식은 함수들을 조립해서 원하는 값을 계산하는 상황을 머릿속에 그리며 프로그래밍한다는 특징이 있다. 즉 문제를 바라보는 시각이 다른 것이다. 이 특이한 '거울관계'는 프로그램의 세계의 범주를 더욱 확장시키고 있는 것이다.

이 '거울관계'가 성립된다는 것은 이런 프로그래밍 언어에 논리를 보태 그 세계가 더욱더 다채로워질 수 있다는 것이다. 기계의 상태변화라는 것은 기계의 등장과 작동원리부터 그러했듯이 결국 데이터가 처리되는 것이 물리적으로 표현되는 것이다. 이는 역으로 기계와 전혀 상관이 없어 보이는 수학적 논리들이 프로그램의 세계를 더 효율적으로 또는 풍요롭게 만들 수 있다는 것이 된다. 책에서 람다 계산법은 이 말을 대변하고 있다. 람다 계산법은 기계적 계산을 식으로 표현할 수 있다. 즉 람다식은 튜링기계와 그 계산 능력이 정확하게 같아서 '기계적인 계산'을 수행할 수 있다.

람다 계산법의 관점은 결국 컴퓨터 세계에서 언어와 논리는 동전의 논리일 뿐 같은 것이라는 결론을 내게 된다. 즉 논리추론의 징검다리들이 컴퓨터 프로그램의 조립 방법과 대응한다는 것이다. 이 '거울관계'는 프로그램의 세계의 범주를 확장시키고 '자연 세계'와 대등한 위치를 점하도록 해준다. 여기에 확률추론 프로그래밍까지 더해지면서 '데이터의 축적'이라는 개념마저 더해져 말그대로 또 하나의 세계를 창조해나간다고 할 수 있을 것이다.