# 컴퓨터 과학이 여는 세계

027.013 Computational Civilization

과제 2

그 도구의 실현 & SW의 세계

경영대학 경영학과

2007- 권오수

Exercise 1 **“그 도구의 실현 & SW의 세계”** 컴퓨터 구현의 원리와 알고리즘에 대한 이야기

# 컴퓨터의 구현 원리, 그리고 무한한 가능성

## 1. 서론

앨런 튜링이 1936년 그의 논문[1]에서 보편 기계의 원리를 제안한 이후 컴퓨터 기술은 경이로운 속도로 발전했다. ‘무어의 법칙(Moore's Law)’은 반도체 마이크로프로세서의 성능이 18개월마다 2배로 증가한다는 사실을 밝히며 컴퓨터 기술 발전을 논할 때에 어김없이 등장하는 상징이 되었다. 뿐만 아니라 소프트웨어 기술 또한 비약적으로 발전하여 아이폰(iPhone)에 탑재된 '시리(Siri)'는 마치 우리의 말을 알아듣는 것 같아 보이기도 하며 윈도우즈나 Mac OS는 과거 텍스트 기반 운영체제에서는 상상도 하지 못했던 직관적인 인터페이스를 갖추고 있다.

이렇듯 하루가 다르게 발전하는 컴퓨터 관련 기술의 발전은 자연스레 우리를 점점 컴퓨터에 대한 ‘이해’와 유리시키고 있다. 점차 복잡해지는 반도체 기술은 우리를 지레 겁먹게 하고, 이와 동시에 직관적이고 편리한 소프트웨어는 그 하부구조에 대한 이해를 요구하지 않기 때문이다. 다시 말해 기술적 고도화와 사용의 직관성 및 단순성이 동반되고 있다.

하지만 컴퓨터의 구현 원리는 어떤 면에서 우리가 현재 사용하는 운영체제들 이상으로 직관적이며 단순하다. 결국 우리가 알고 있듯이 컴퓨터는 1과 0, 혹은 YES와 No 등 어떻게 표현이 되던 간에 두 개의 상태로 이루어진 논리 기능을 기반으로 연산을 수행하며 마이크로프로세서는 이를 구현해낸 것이기 때문이다.

또한 우리가 0과 1만으로 컴퓨터를 다루지 않고, 또 복잡한 연산을 수행할 수 있는 이유는 기능적 추상화(functional abstraction) 때문이다. 이는 컴퓨터의 하드웨어의 단순한 논리 기능이 복잡한 임무를 수행하고, 또 그 하드웨어를 제어하는 기계어부터 상위의 프로그래밍 언어와 운영체제에 이르기까지 컴퓨터의 계층적 구조를 설명할 수 있는 개념이다. 이 기능적 추상화로 인해 보다 상위 레벨의 프로그래밍뿐만 아니라 지금 하고 있는 문서 작업을 직관적으로 할 수 있는 것이다.

현존하는 모든 컴퓨터는 튜링 기계이다. 모든 컴퓨터는 동일한 원리로 구현되고 있으며 보편 컴퓨터의 원리라고 하기도 한다. 이는 다시 말하자면 우리가 앞서 언급한 컴퓨터의 구현 원리를 딱 한 번만 제대로 이해한다면 현존하는 모든 컴퓨터를 이해할 수 있다는 뜻이며 그 컴퓨터들의 한계점 또한 느낄 수 있다는 것이다. 본 보고서에서는 컴퓨터의 구현 원리와 그 한계점을 이해함으로써 앞으로의 가능성을 논의해 보고자 한다.

## 2. 본론

### 2.1 복잡한 명제가 전기 회로에 구현되어 처리되기 까지

컴퓨터의 구현 원리 이해를 위해서는 가장 먼저 어떻게 복잡한 명제들이 전기 회로 상에서 표현 및 처리되는지 살펴볼 필요가 있다. 조지 불은 불 대수를 통해 복잡한 명제의 참과 거짓을 대수적으로 연산할 수 있는 방법을 고안했다. 이는 AND, OR, 그리고 NOT을 통해 명제를 표현하고 또 대수적으로 연산가능하게 했다는 점에서 기호 논리학의 기념비적 발전으로 평가된다.

섀넌은 1940년 그의 박사 논문 「계전식 스위치 회로의 기호 해석」에서 불 대수의 표현을

1) 「ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM」, Alan M. Turing, 1936.

전기 회로에 그대로 옮길 수 있다는 사실을 증명했다. 회로에서 스위치가 닫혀 회로가 연결되면 명제가 참인 것을 의미하고, 스위치가 열려 회로가 끊긴 것은 거짓을 의미했다. 더 나아가 AND는 두 개의 스위치의 직렬연결로, OR은 병렬연결로, 그리고 NOT은 뒤집기로 구현이 가능했기 때문에 불 대수로 표현된 논리식을 그대로 전기 회로에 구현할 수 있었던 것이다.

이 두 가지 발견은 1) 어떤 임무라도 논리 기능으로 변환될 수 있고, 2) 그 논리 기능을 회로와 스위치의 연결을 통해 실제 구현할 수 있다는 점에서 컴퓨터의 토대를 이루었다고 할 수 있었다. 비록 인간의 사고 과정이 정확하게 불 대수의 논리 처리 방식으로 이루어지는지 여부는 현재까지 알 수 없다고 하지만, 이러한 방법을 통해 복잡한 명제를 처리하고 임무를 수행하는데 있어서는 매우 유사한 수준에 도달했다고 볼 수 있다.

하지만 논리 기능의 구현이 반드시 전기 회로로만 이루어질 수 있는 것은 아니다. 수압 밸브, 화학 반응, 그리고 기계식으로도 가능하다. 결국 0과 1, 전기가 흐르거나 그렇지 않은 상태와 같이 두 가지 상태만 존재하는 '비트'의 형태로 신호를 전달할 수만 있다면 어떤 방식으로든 논리 기능의 구현을 통해 컴퓨터를 만들 수 있는 것이다. 실제로 파스칼, 라이프니츠와 같이 유명한 과학자들이 발명해 낸 초기의 계산 장치들은 모두 기계적 논리 장치라고도 볼 수 있다.

비록 논리 기능을 다양한 방법으로 구현해낼 수 있다고 하지만, 실제 완전한 컴퓨터로써 복잡한 연산을 수행하기 위해서는 몇 가지 조건이 충족되어야 한다. 먼저 입력 신호가 출력 신호에만 영향을 주고 그 반대 현상은 일어나지 않는 비대칭 스위치가 있어야 한다. 또한 신호의 증폭 기능도 수행할 수 있어야 한다. 이는 입력 신호가 감쇠되거나 왜곡되더라도 출력 신호는 언제나 최대치가 되어야 한다는 뜻으로 "복원 논리(restoring logic)" 라고도 한다. 마지막으로 커넥터를 통해 출력 신호를 분기하여 여러 개의 또 다른 입력 단에 접속할 수 있어야 한다. 그래야만 복잡한 논리식을 표현해 낼 수 있기 때문이다. 이 경우에도 복원 논리를 통해 분기 과정에서 약해진 신호를 최대로 증폭할 수 있어야만 한다.

위의 세 가지가 보장될 때 수천 단계의 논리 블록이 연결되어도 오류 없는 처리를 가정할 수 있다. 이러한 조건만 만족시킨다면 크기를 줄일 수 있다는 점 빼고는 반도체 기술로만 컴퓨터를 만들어야 하는 특별한 이유는 없다. 어떤 기술을 쓰든 스위치(제어요소)와 커넥터(연결요소)라는 두 가지 요소와 복원논리만 있다면 컴퓨터를 만들 수 있는 것이다.

## 2.2 유한 상태 기계와 프로그램

그렇다면 앞서 살펴본 원칙들이 어떻게 컴퓨터로 구현되는가? 다소 원론적인 이야기에서 더 나아가 보다 구체적으로 살펴볼 필요가 있다. 단도직입적으로 표현한다면 불 대수의 OR과 NOT연산만 있다면 어떤 논리 블록도 구현이 가능하다. 가장 먼저 AND는 OR과 NOT을 조합하여 만들어낼 수 있으며 AND, OR, 그리고 NOT을 조합하면 다른 연산도 가능해진다.

예를 들어 다수결 판정 함수는 어떨까? A, B, C 세 사람이 찬성(1) 혹은 반대(0)를 할 수 있고 찬성이 많을 경우 1을, 반대가 많을 경우 0을 출력하는 함수는 논리 기능을 통해 쉽게 구현할 수 있다. 이는 세 사람이 찬성하거나, 셋 중 두 사람이 찬성하는 경우에 1을 출력하면 되므로 $\{(ABC)+(\overline{A}BC)+(A\overline{B}C)+(AB\overline{C})\}$를 연산하면 된다. 또한 이를 전기 회로로 구현하는 것은 그리 어렵지 않은 일이다. 이처럼 불 대수를 활용하여 구현한 논리 블록은 컴퓨터 연상 장치의 토대가 된다.

더 나아가 현재의 입력만이 아니라 이전의 입력 값에도 영향을 받게 할 경우 보다 복잡한 연산을 수행할 수 있게 되는데 이를 유한 상태 기계라고 한다. 이전의 입력 값을 반영하기 위해 유한 상태 기계는 레지스터(상태 블록)를 논리 블록에 연결하여 논리 블록의 출력을 레지스터에

적는다. 이를 '상태'라고 하고, 논리 블록은 입력과 상태를 근거로 출력 값과 다음 상태를 계산해 내는 것이다. 이 과정에서 레지스터는 현재 상태를 '기억'하고 있어야만 하는데 이는 "flip-flop"이라는 회로를 통해서 구현할 수 있다. 이 역시 불 논리 블록으로 구현 가능하다.

위에서는 다수결 판정 함수를 전기 회로에 곧바로 구현하여 값을 얻고자 했다면 튜링이 제안한 유한 상태 기계에 명령을 내리는 일을 보다 보편적인 일이다. 다시 말해 튜링 머신은 어떤 프로그램을 수행하는 보편 기계이며 프로그램을 통해 명령을 받고 수행한다. 그렇다면 프로그램은 무엇이고 유한 상태 기계와는 어떻게 관련을 맺는가?

프로그래밍은 컴퓨터가 수행해주었으면 하는 일을 설명하는 작업이다. 이 과정에서 자신만의 문장 구조를 가지고 있는 프로그래밍 언어를 활용한다. 뿐만 아니라 언어 내에서 기능적 추상화의 일환으로 새로운 언어를 정의할 수 있는데 이를 원시 언어를 이용하여 정의한 '서브루틴(subroutine)'이라고 한다.

유한 상태 기계와 프로그램을 연결하기 위해서는 가장 먼저 유한 상태 기계에 '메모리'를 덧붙여야 한다. 메모리에는 요청받은 임무에 관한 정의를 저장하는 장치인데 마찬가지로 불 논리 블록으로 구현가능하다. 다음으로 유한 상태 기계를 작동시킬 때 필요한 원시 언어인 기계어로 적힌 명령을 수행하게 되는데, 이 과정에서 프로그래밍 언어를 기계어로 해석(interpret)하는 과정이 필요하다. 이렇게 메모리를 덧붙인 유한 상태 기계는 아래의 연산을 반복적으로 수행한다.

| 처리명령 | | | 제어명령 |
|---|---|---|---|
| 메모리에서 나온 명령 읽기 | → | 명령에 따른<br>연산 수행 및 값 출력 | → 다음 명령의 주소 계산 |

## 2.3 기능적 추상화

기능적 추상화는 2.1에서 서술한 단순한 논리 기능의 구현이 모여 복잡한 연산을 수행을 가능하게 하고 더 나아가 보다 상위 레벨의 프로그래밍을 가능하게 한다. 사용자 측면에서는 현재 우리가 직관적으로 사용하는 운영체제 기반 PC와 스마트폰에 이르기까지 복잡한 하부구조에 신경 쓰지 않고 필요한 작업을 할 수 있게 한다. 위키피디아에서는 추상화를 다음과 같이 정의하고 있다.

> *In computer science, abstraction is the process by which data and programs are defined with a representation similar in form to its meaning (semantics), while hiding away the implementation details. Abstraction tries to reduce and factor out details so that the programmer can focus on a few concepts at a time. A system can have several abstraction layers whereby different meanings and amounts of detail are exposed to the programmer. For example, low-level abstraction layers expose details of the computer hardware where the program is run, while high-level layers deal with the business logic of the program.*[2)]

위의 정의에 따르면 기능적 추상화는 데이터와 프로그램을 실제 처리하는데 필요한 하부의 구체적인 사항들은 감추고, 의미를 중심으로 정의하는 일련의 과정이다. 하나의 시스템은 하드웨어부터 실제 사용자 인터페이스에 이르기까지 여러 개의 추상화 계층으로 이루어지는데 이를 통해 각 계층상의 복잡성을 줄이고 작업에 필요한 몇 가지 개념에만 집중할 수 있게 돕는다.

---

2) 'Abstraction(computer science)', Wikipedia, http://en.wikipedia.org/wiki/Abstraction_(computer_science)

이처럼 기능적 추상화와 계층 구조는 복잡한 시스템을 이해하고 활용하기 위한 가장 강력한 도구라고 할 수 있다. 전기 회로로 구현된 단순한 논리 기능은 무수한 논리 기능들과 결합됨으로써 복잡한 연산을 수행할 수 있게 되는데 이는 반도체 회로를 현미경으로 보면 알 수 있듯이 굉장히 복잡한 과정이다. 하지만 기능적 추상화에 따르면 단순한 논리 구조나 이들의 합이나 근본적인 원리에서는 다를 것이 없으며 일단 올바로 정의된 이후 그 상위 계층에서는 이러한 내용을 신경 쓸 필요가 없는 것이다.

보다 구체적인 예를 들어보면 컴퓨터 프로그램은 영어를 활용한 의미 중심의 프로그래밍 언어로 작성되는데 이 언어를 운영 체제가 기계어 명령열로 변환한다. 컴퓨터의 메모리에 저장되는 그 명령들을 유한 상태 기계가 호출하여 실행한다. 유한 상태 기계는 결국 불 논리의 함수를 물리적으로 구현한 전기 회로인데 이곳에서 신호가 전달되며 명령이 처리되는 것이다.

반면 기능적 추상화는 복잡하지만 견고한 하위 계층을 바탕으로 또 하나의 복잡한 상위 계층을 만들어 내는데 기여하므로 결과적으로 우리의 이해나 통제를 벗어나는 수준의 복잡도를 만들어내는 역설을 내포하고 있다. 따라서 우리는 기능적 추상화의 이기를 누림과 동시에 직관적인 이해를 위해 전기식 컴퓨터를 기계적인 관점으로 환원해보는 등의 노력이 필요하다.

## 2.4 보편 기계와 가능성

튜링 기계는 다른 말로 보편 기계라고 하기도 한다. 그의 주장에 따르면 모든 컴퓨터는 레지스터와 불 논리로만 이루어져있으며 임의의 두 컴퓨터 사이의 성능 차이는 속도와 메모리 크기 두 가지 밖에 없다. 이를 증명하기 위해 다음의 세 가지 측면에서 보편 기계를 검증해보자 1) 이진 컴퓨터 측면 2) 계산 불능 문제 측면 3) 무작위성

가장 먼저 이진 컴퓨터 측면에서 우리는 이러한 상상을 할 수 있다. 만약 0과 1이 아닌, 세 가지 상태를 갖는 비트를 기반으로 정보를 처리할 수 있다면 어떨까? 결론을 먼저 이야기하면 삼진 컴퓨터는 크기가 작아지고 제조 단가 또한 낮아지겠지만, 이진 컴퓨터에서 비트를 두 개 사용한 것과 같다. 마찬가지로 무한개의 상태를 갖는 아날로그 신호 또한 비트를 몇 개 추가함으로써 모방할 수 있으며 이는 정확도가 떨어지는 아날로그에 비해 심지어 더 나은 방법이다.

계산 불능 문제 측면에서는 어떨까? 사실 올바르게 정의되고도 계산 불능인 문제는 거의 없다고 한다. 예외가 있다면 튜링이 제시한 '정지 문제(halting problem)'가 대표적인데 이는 컴퓨터의 탓이 아니라 원래 해결이 불가능 한 것이라는 주장이 제기되고 있다. 또 하나의 예외인 '수학 명제의 참 거짓 판단 문제'는 인간 또한 해결할 수 없다는 측면에서 컴퓨터에 대한 필요 이상의 가혹한 비판이라는 의견이 존재한다.

마지막으로 무작위성 문제는 컴퓨터가 완전무결한 무작위 숫자를 만들어낼 수 없다는 점에서 기인한다. 컴퓨터의 난수 생성 능력은 유사 난수 혹은 카오스 시스템, 즉, 인풋이 조금만 변해도 최종 출력이 크게 변화하는 정도에 머물기 때문이다. 하지만 최근 양자 역학 기술을 활용한 컴퓨터의 가능성이 제기되면서 이 문제를 해결할 가능성이 열리고 있다.

양자 역학 기술은 양자의 동시 계산 능력과 양자 얽힘 현상을 활용하여 병렬 처리를 가능케 하며, 하나의 양자 논리 블록이 수행할 수 있는 계산량을 거의 무한한 수준으로 끌어올릴 수 있는 가능성이 있다. 하지만 '결 어긋남'이라고 불리는 현실적 제약으로 인해 실제 구현을 위해서는 아직 연구가 필요한 시점이다.

## 2.5 알고리즘과 휴리스틱

그렇다면 이 세상에 단 한 가지의 형태로 존재하는 컴퓨터의 구현 원리에 대해 알았으니, 기

능적 추상화의 논리에 따라 보다 상위의 수준에서 실제 컴퓨터로 문제를 해결할 때 활용하는 절차에 대해 살펴볼 필요가 있다. 컴퓨터 프로그램은 문제를 풀기 위한 논리적 절차인 알고리즘과 휴리스틱의 일종이라고 볼 수 있는데 이 둘은 무엇이며 또 어떻게 다를까?

알고리즘은 "어떠한 문제를 해결하기 위한 여러 동작들의 유한한 모임"으로 정확성, 작업량, 기억 장소 사용량, 최적성, 그리고 복잡성을 기준으로 분석한다.[3] 또한 분석 시 하나의 특정 문제 해결을 기준으로 하는 것이 아닌 결합되는 요소가 증가할 때 한계적으로 증가하는 계산 속도 및 기억 장소 사용량을 주요하게 여긴다.

어떤 알고리즘이 해당 문제를 해결하는 최적의 방법인지 판단하는 것은 매우 어려운 일이다. 이는 바로 앞서 언급한 것처럼 결합적 폭발(combinational explosion), 즉, 결합되는 요소들의 수가 많아짐에 따라 경우의 수가 지수적으로 커질 경우 증가하는 복잡도를 감당해내기가 쉽지 않기 때문이다.

이런 이유로 실제 문제를 해결하는데 있어서는 알고리즘이 아닌 휴리스틱이 더 많이 활용된다. 휴리스틱은 최선이 아닌 합격점 수준의 규칙을 의미하는데 속도가 느리거나 종종 완벽한 문제 해결이 불가능한 알고리즘보다 우수한 휴리스틱이 현실적 대안으로써 더 나을 때가 많다.

## 3. 결론

『생각하는 기계』의 저자 대니얼 힐리스는 같은 책에서 다음과 같이 말하고 있다.

> *"이는 프로그램만 적절히 만들어 주면 인간의 뇌가 하는 일도 보편 컴퓨터가 따라서 할 수 있다는 것을 의미한다"*[4]

원칙적으로 모든 컴퓨터가 동일한 원리를 바탕으로 한 튜링 기계라고 한다면, 현재의 튜링 기계로 위와 같은 일이 가능할까? 아니면 전혀 새로운 원리가 고안 및 적용되어야 인간의 뇌를 모방할 수 있는 수준에 도달할까?

컴퓨터와 인간의 뇌를 구분 짓는 가장 큰 차이는 학습과 이를 통한 추상적인 개념의 습득이 아닌가 싶다. 학문적인 엄밀성을 차치하고, 직관적으로 생각해보았을 때 인간은 튜링머신처럼 한 번에 단 하나의 정보만 처리하는 것은 아니다. 인간은 순식간에 오감 모두로 부터 정보를 제공받아 처리하고 있으며 이를 통해 끊임없이 학습한다. 흔히 걸음마를 배울 때를 예로 들지만 최근 중국어를 학습하고 있는 나는 이를 간접 체험하고 있다. 처음에는 잘 만들어지지 않던 문장들이 이제는 조금씩 입에서 나오는 것을 보며 인간의 학습 능력의 경이로움을 느끼고 있다.

최근에는 멀티 코어 프로세서들을 스마트폰에서도 찾아볼 수 있듯 병렬 컴퓨팅 기술이 발전하며 컴퓨터의 직렬 처리 문제가 해결되고 있다. 뿐만 아니라 양자 컴퓨팅 기술이 현실화된다면 정말 인간의 처리 능력에 가까워질지도 모르겠다.

하지만 처리 능력과 학습능력은 전혀 다른 문제다. 어쩌면 여기서부터는 소프트웨어의 영역이 아닐까 조심스럽게 추측해 본다. 대니얼 힐리스가 시뮬레이션 진화 기법을 통해 만든 정렬 프로그램[5]이 직접 작성한 어떤 프로그램보다 빨랐다는 사실, 그리고 알고리즘을 전혀 이해할 수 없었다는 사실은 마치 이해할 수 없지만 경이롭기 그지없는 우리의 뇌를 연상시킨다. 소프트웨어가 이렇게 고등 학습 능력을 만들어 낼 수 있다면? 앞으로 소프트웨어가 나을 지능의 압축적 '진화과정'을 떠올려보자, 설레지 않는가!

---

3) '알고리즘', Wikipedia, http://ko.wikipedia.org/wiki/알고리즘
4) 『생각하는 기계』,대니얼 힐리스, 사이언스 북스, 2006, 119p
5) 『생각하는 기계』,대니얼 힐리스, 사이언스 북스, 2006, 250p