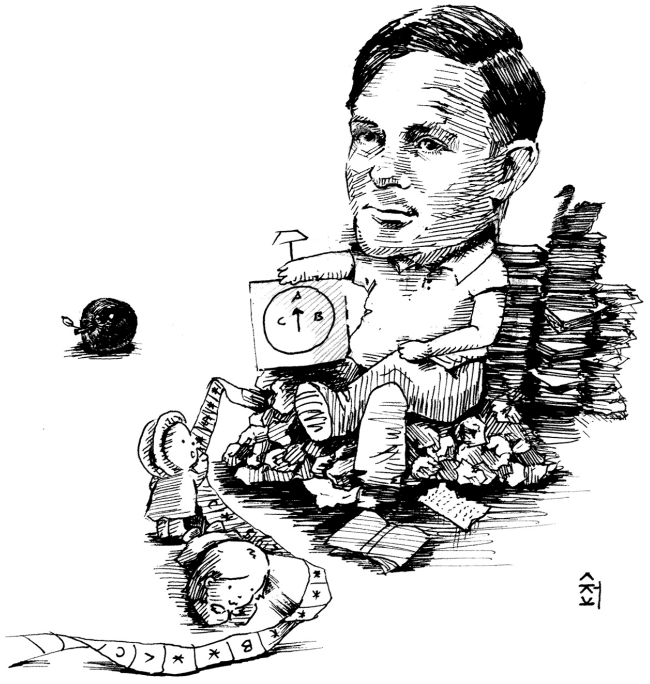
# 2 장

# 400년의 축적





컴퓨터과학이 여는 세계 ⓒ이광근 2015

컴퓨터는 특별하다. 인류역사에 유례가 없던 도구다. 다른 도구들과는 근본적으로 다르다. 칼, 바퀴, 냉장고, 자동차 등을 보자. 칼은 자르는 데만 쓰고, 바퀴는 구르는 데만 쓴다. 냉장고는 식히는 데만 쓰고, 자동차는 이동시키는 데만 쓴다. 반면에 컴퓨터는 쓸모가 많다. 한이 없다.

컴퓨터 하나로 할 수 있는 일을 보자. 컴퓨터로 문서를 편집하고, 인터넷도 하고, 동영상도 본다. 게임도 하고, 쇼핑도 한다. 소문도 퍼뜨리고, 전화도 건다. 또 자동차도 운전하고, 로켓도 날린다. 로봇도 운전하고, 핵발전소도 돌린다. 모두 컴퓨터 하나로 할 수 있는 일들이다. 그래서 컴퓨터는 *보편만능의 기계*$_{universal\ machine}$라고 부른다.

누가 이런 놀라운 도구를 발명한 걸까? 그 답은 인류의 대담한 꿈과 좌절의 드라마로 펼쳐진다. 좌절의 아이러니, 예상 밖의 검은백조.

## 2.1 탄생

"보편만능의 도구", 그것 하나면 어떤 일이든 할 수 있는 기계. 이것은 어떻게 탄생한 걸까?

의외였다. 컴퓨터는 20세기 수학자들의 큰 꿈이 철저히 좌절되는 과정에서 나온 부산물이었다. 좌절을 엄밀히 확인하는 데 고안된 소품이 21세기 정보혁명의 주인공이 된다.





### 2.1.1 청년

1936년, 손기정 선수가 베를린 올림픽 마라톤에서 우승하던 바로 그 해였다. 마라톤 경기(8월 9일)를 2달 정도 남긴 5월 28일.

손기정

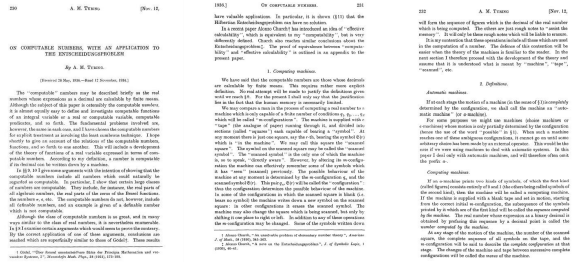

앨런 튜링

*아들아이는 범이라고 장고기를 잘 잡는 앞니가 빠드러진 나와 동갑이었다*

- 백석, [주막]

손기정 선수와 동갑내기인 영국의 앨런 튜링(Alan Turing)이라는 청년. 그는 다음 제목의 논문을 런던 수리학회(London Mathematical Society)에 제출한다.

> *"계산가능한 수에 대해서, 수리명제 자동생성 문제에 응용하면서"(On Computable Numbers, with an Application to the Entscheidungsproblem)*



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

...

이 논문에서 튜링은 컴퓨터의 근본적인 디자인을 최초로 선 보인다. 이 때 튜링은 2년 전 캠브릿지 대학 수학 학부과정을 최우수 성적으로 마친 24세의 청년이었다. 그 해 9월 프린스턴 대학(Princeton University) 수학과 알론조 처치(Alonzo Church)교수의 지도를 받기위해 유학떠나기 직전이었다.

### 2.1.2 소품

그런데 제목만으로는 도저히 컴퓨터와 관계없을 것 같은데, 어떻게 그 논문에 컴퓨터의 원천 설계도가 나타난 걸까? 컴퓨터의 모델을 "튜링기계(Turing Machine)"라고 부르고, 컴퓨터분야의 노벨상을 "튜링상(Turing Award)"라고 부르며 "튜링"이라는 이름이 항상 붙는 연유는 뭘까?

영 이상하지 않은가? 컴퓨터라는 비범한 도구를 최초로 설계한 논문의 제목이란게. *"계산가능한 수에 대해서, 수리명제 자동생성 문제에 응용하면서"* 라니?

재미있게도, 위의 논문은 "이러한 특이한 도구를 디자인 했으니 보라"고 주장한 논문은 아니다. 위의 논문은 그 당시 수학계에 내리친 청천벽력의 좌절을 전해들은 튜링이 색다른 방식으로 그 사실을 다시 증명해 본 것이다.

그런데 이 논문에서 컴퓨터의 원천 설계도가 슬그머니 드러난다. 아이러니하게도, 좌절을 증명하는 작품속에 인류의 정보혁명을 이끌 도구의 설계도가 주요 소품으로 등장했던 것이다.

튜링이 리바이브한 당시 수학계에 내리친 청천벽력은 쿠르트 괴델(Kurt Gödel)이 1931년에 증명한 다음의 사실이었다.

*"기계적인 방식만으론 수학의 모든 사실들을 만들어 낼 수 없다."*

### 2.1.3 꿈

1928년. 미키 마우스가 세상에 데뷔한 해. 대담한 꿈이 유럽 수학계에 번지고 있었다. 당대 선두의 수학자였던 다비드 힐베르트(David Hilbert)가 독려한 꿈이었다.





그의 생각은 이렇다. 수학자들이 해왔던 작업과정을 보아하니, 몇개의 생각의 법칙(추론 규칙)을 반복 적용하는 게 다인 듯 했다. 혹시, 사실 아닐까? 몇개의 추론규칙만 가지면 앞으로 수학자들이 증명할 모든 명제들이 모두 술술 찾아지는 게 아닐까?

"모두"와 "술술"이 핵심이다. "모두"란 말 그대로 "하나도 빠뜨리지 말고"라는거고, "술술"이란 "쉽게 자동으로" 라는 뜻인터, 그런 추론규칙만 찾아진다면 수학자가 더 이상 고생할 필요가 없어지는 것이다. 모든 수리명제가 자동으로 술술 찾아질 것이다.

그런데, 자동으로 적용할 수 있는 추론규칙이란 어떤 것을 말하는 걸까? 예를 들어 이런 것이다. 셜록홈즈가 각 사건을 해결하는 과정을 보자. 모든 사건마다 드라마의 전개 내용은 제 각각이다. 하지만 셜록홈즈가 매번 사건을 해결하는 추론 과정을 일반화해서 보면 같은 패턴의 과정을 반복할 뿐이다. "A이면 반드시 B이다, 그리고 지금 A가 사실이다, 그렇다면 B가 사실이어야 한다," "A가 사실이라면 지금 상황이 말이 안된다, 그렇다면 A는 사실이 아니다" 등의 것들이다. 이런 것이 자동으로 적용할 수 있는 추론규칙(패턴)이다. 셜록홈즈는 사건마다 "A"와 "B"에 구체적인 경우를 대입해서 새로운 사실을 유추해내고 이 때 하는 일이란 지금까지 알려진 사실들을 대입할 수 있는 추론규칙을 찾는 것이다. 이 스텝은 자동화 할 수 있는 단순 작업이다. 추론규칙의 전제("그렇다면"의 왼쪽)에 현재 알고 있는 사실을 대입해보면, 그 규칙의 결론("그렇다면"의 오른쪽)에 해당하는 사실이 드러난다. 예를들어 위의 두번째 추론규칙을 적용하는 예다. "장그래 박사가 그저께 사망했다면 어제 세미나에 참여했는데 말이 않된다" 그렇다면 "장그래 박사는 그저께 사망하지 않았다." 이렇게 적용가능한 추론규칙을 동원해서 새로 알게되는 사실들을 모두 모아갈 수 있다. 새로 알게된 사실때문에 또 다른 새로운 사실을 알게된다. 이렇게 연쇄반응 하듯이 새로운 사실을 생산해내는 엔진은 셜록홈즈의 머리속에 있는 몇 가지의 고정된 추론규칙(패턴)들이다.

힐베르트가 제안한 문제가 수학자들이 사용하는 이런 규칙들을 찾아보자는 것이다. 한없이 많지는 않을 것이므로 찾아보자. 힐베르트는 이 문제를 1928년 국제수학자대회에서 공식적으로 제기한다. 기계적인 방식으로 – 자동

으로 – 모든 수학의 사실들을 술술 만들어 낼 수 있을 것 같다는 꿈, 그래서 수학자의 고된 짐을 벗을 수 있을 것 같다는 꿈이었다.

### 2.1.4 좌절

하지만 3년후 그 꿈은 산산히 조각난다. 1931년 쿠르트 괴델(Kurt Gödel)이라는 25세의 신참 수학자였다.

괴델이 내려친 벼락은 "꿈 깨시오"였다. 이 꿈은 절대 이루어질 수 없다고 증명해버린 것이다. 기계적인 방법만으로는 수학에서의 사실을 모두 샅샅히 만들어낼 수 없다는 것이다. 기계적인 방식만으론 참인지 거짓인지 판단할 수 없는 명제가 항상 존재한다는 증명이었다. 그래서, 수학자들이 고생해서 알아내는 사실들을 자동으로 척척 빠뜨리지않고 모두 만들어내는 기계는 불가능하다는 것이었다.

*"기계적인 방식만으론 수학의 모든 사실들을 길어올릴 수 없다."*

아무리 추론규칙(생각의 법칙)들을 잘 만들어도, 그 법칙들만으로는 모든 참인 명제들을 길어올릴 수 없다. 만든 그물을 빠져나가는 사실들은 반드시 있다. 놓친 사실까지 길어올리도록 매번 기계를 확장한다해도, 그 기계가 또 길어올리지 못하는 사실이 항상 존재한다. 그물을 기워도 기워도 뚫린 데가 항상 나타난다.

*불완전성 정리*$_{incompleteness\ theorem}$라고 불리는 그 증명은 당대 수학계를 발칵 뒤집어놓는다. 괴델의 증명을 확인하고 재확인하는 세미나와 강의가 캠퍼스의 고요한 대낮을 끓게하였고, 그 증명은 거부할 수 없는 사실임이 모두에게 신속히 확인된다.

### 2.1.5 캠브릿지

1935년. 캠브릿지대학(University of Cambridge)의 수학과를 갓 졸업한 알랜 튜링(Alan Turing)은 괴델의 증명 소식을 듣는다. 20대 초반의 청년이었다. 졸업후 한 강의를 통해서 였는데, 38세의 수학과 막스 뉴만(Max Newman)교수가





개설한 강의였다. 강의내용은 괴델의 *불완전성 정리*$_{incompleteness\ theorem}$의 증명을 리뷰하는 것이었다.

튜링은 이 강의를 듣고 괴델의 증명을 단도직입적으로 다시 증명해 볼 수 있겠다는 생각을 한다. 그리고 똑 같은 증명을 자신만의 방식으로 진행한다. 이 증명을 정리한 것이 위에 소개한 1936년의 논문이다.

튜링은 이 증명에서 간단한 기계부품들을 정의한다. 그리곤 이 부품들로 하나의 특별한 기계를 만들어 보이는데, 그 기계가 바로 *보편만능의 기계*$_{universal\ machine}$, 즉 컴퓨터였던 것이다.

### 2.1.6 단도직입

우선, 튜링은 그 증명에서 왜 기계부품을 정의할 필요가 있었을까? 튜링의 증명은 단도직입적이다. "기계적인 방식"이 뭔지를 곧바로 정의한다. 그리고 그 방식만으론 모든 사실들을 만들 수 없음을 증명한다.

튜링은 아주 단순한 다섯 종류의 기계 부품을 정의한다. 마치 레고 블럭 셋트같은 것들이다. 그리고 그 부품들로 만든 기계로 돌릴 수 있는 것만을 "기계적인 방식"이라고 정의한다. 그러고는 이 방식으로는 절대 돌릴 수 없는 계산문제를 하나 보인다. 그리고 이 문제를 지렛대 삼아 기계적인 방식으로는 참인 사실을 하나도 빠뜨리지 않고 만들어 내는 것은 불가능하다는 결론을 이끌어 낸다.

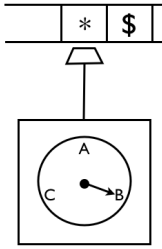
이 증명의 중앙부 가파른 협곡을 통과하면서 튜링은 컴퓨터의 설계도를 펼쳐보이게된다. 튜링은 우선 자신이 정의한 "기계적인 방식"이 충분히 광범위하다는 것을 설득해야 했다. 그래서 튜링은 자기의 방식으로 돌아가는 기계들의 예를 보여주기 시작한다. 그리고는 그 정점의 예로, 하나의 특이한 "궁극의 기계"를 만들어 보인다. 이 정도의 기계까지 만들어 낼 수 있는 기계부품들은 그러므로 충분히 광범위한게 아니냐는 걸 암시하면서.

그리곤 그 "궁극의 기계"를 타고 증명의 중심부 협곡을 아슬아슬 통과해 간다. 이 궁극의 기계를 이용해서, 참인 명제를 모두 만들어주는 기계가 불가능하다는 것을 보이는 데, 바로 이 궁극의 기계에 컴퓨터의 원천 설계도가 담기

게 된다.

### 2.1.7 기계

이 궁극의 기계를 소개하기 전에 우선 튜링이 정의한 기계부품들이 무엇인지 살펴보자. 그 기계부품들은 너무나 단순하다. 아래가 그 부품으로 만든 튜링 기계의 한 예다:




| 현상태 | 읽은심볼 | 쓸심볼 | 다음칸 | 다음상태 |
|---|---|---|---|---|
| A | * | * | > | A |
| A | $ | $ | > | B |
| B | * | * | < | C |
| C | $ | * | > | C |
| C | * | $ | > | B |

부품들은 몇 개 안된다. 무한히 많은 칸을 가진 테잎, 테잎에 기록되는 심볼들($, * 등 유한 개), 테잎에 기록된 심볼을 읽거나 쓰는 장치, 그 장치의 상태를 나타내는 심볼들(A, B, C 등 유한 개), 그리고 기계의 작동규칙표다. 기계마다 이 부품들이 정해진다. 어떤 심볼들을 테잎에 읽고 쓰게 될 것인지, 어떤 심볼들로 상태를 구분할지, 작동규칙표는 무엇인지. 그리고 테잎의 시작 모습과 기계의 시작 상태 심볼, 그리고 테잎에서의 시작 위치가 정해진다. 그림에서 박스 안에 시계 바늘이 현재의 상태 심볼을 가리킨다.

이렇게 정의된 기계는 작동규칙표에 적힌대로 작동한다. 튜링기계가 하는 일은 단순하다. 테잎 칸의 심볼을 읽고 쓰면서 테잎위를 기껏해야 한 칸씩 좌우로 움직여 가는 일만 할 수 있다. 이러면서 기계의 상황이 매번 변경된다.

위 그림에서 작동규칙표의 한 행이 하나의 작동 규칙이다. 작동 규칙의 의미는 간단하다. 기계가 어떤 상태에서 어떤 심볼을 읽으면 어떤 심볼을 쓰고, 읽는 장치는 어느방향으로 움직이며(오른쪽 >, 왼쪽 <, 제자리 |), 새로운 기계



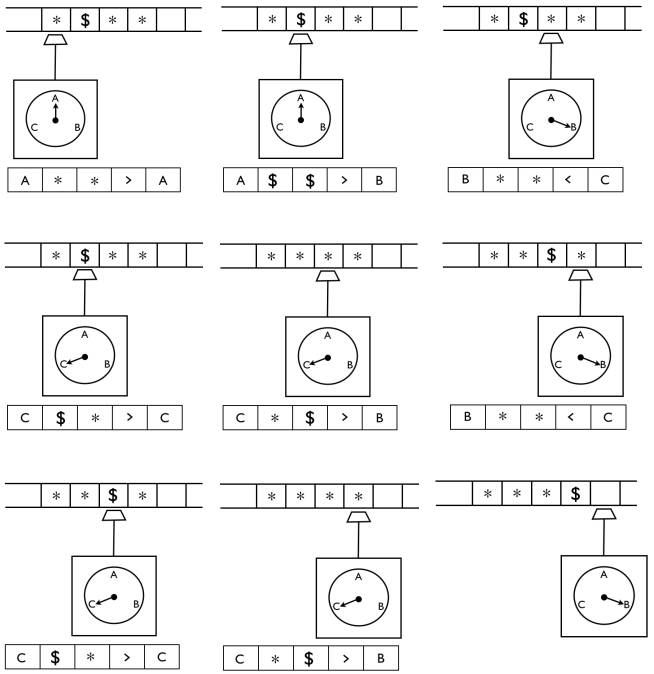


상태는 무엇이 되는지가 표현되어 있다. 예를 들어 아래 규칙이 주문하는 일이란,

| 현상태 | 읽은심볼 | 쓸심볼 | 다음칸 | 다음상태 |
|---|---|---|---|---|
| C | * | $ | > | B |

"현재 상태가 C이고 테잎에서 읽은 심볼이 *이면 읽은 곳에 $를 덮어쓰고, 읽고쓰는 장치를 오른쪽으로 한 칸 움직이고(>), 다음 상태는 B가 되라"는 것이다.

위에 보인 기계가 작동하는 모습은 아래와 같다. 각 상황의 아래쪽에 있는 것이 그 상황에서 적용하게 될 작동규칙이다. 윗줄부터 왼쪽에서 오른쪽으로 진행한다.



위의 기계가 하는 일은 테잎에 있는 $ 심볼을 오른쪽 끝으로 옮기는 것이다. 우선(A상태) $ 심볼을 찾아 오른쪽으로 간다. 찾았으면(B상태) 그게 오른쪽 끝인지를 확인한다. 끝이 아니면(C상태) 옮겨야 하므로, $를 오른쪽으로 한 칸 옮겨놓는다. 한 칸 옮기고는 오른쪽 끝이 아니면 계속 옮겨야 하므로, B상태로 돌아가서 반복하면 된다. 옮기고 나서 그게 오른쪽 끝이라면(마지막 상황) 해당하는 규칙이 없으므로 작동이 멈춘다.

튜링은 이렇게 정의한 기계부품들로 다양한 일을 하는 기계들을 만들어 보이면서, "기계적인 계산"이란 그렇게 만든 기계로 돌릴 수 있는 것들로 한정해도 충분한 것 같다고 설득한다. 테잎에 0과 1을 반복해서 쓰는($010101\cdots$) 기계에서 부터, 0을 시작으로 1을 점점 많이 쓰는($0010110111\cdots$) 기계 등을 만들어 보인다. 사칙연산을 하는 기계도 물론 만들 수 있다 (예를 들어, 2+3을 하기위해 테잎에 1진법으로 $*11*111*$를 써넣고 더하기 튜링기계를 작동 시키면 테잎에 5에 해당하는 1진수를 그 옆에 쓴다: $*11*111*11111*$).

### 2.1.8 궁극

튜링이 만들어 보인 "궁극의 기계"도 이렇게 구성되는 튜링기계의 하나이고 이 기계에 컴퓨터의 핵심능력이 처음으로 등장한다. 컴퓨터의 핵심 능력은 하나의 컴퓨터가 모든 일을 할 수 있다는 점이다. "궁극의 기계"가 바로 그 능력을 처음으로 보여준 설계인 것이다.

이 궁극의 기계는 두 개의 대담한 설계를 담고있다. 첫째는, 그 기계의 정해진 테잎 심볼들만 가지고 임의의 튜링기계를(작동규칙표와 테잎 그리고 현재 기계 상태 심볼)를 1차원 실로 – 일렬로 늘어선 심볼들로 – 표현할 수 있다는 점이다. 그래서 임의의 다른 기계의 정의를 테잎에 입력으로 받을 수 있게 된다. 둘째는, 테잎에 표현된 기계의 정의에따라 그 동작을 그대로 흉내낼 수 있도록 작동규칙표를 정의한 점이다. 그래서 자기의 고정된 작동규칙표를 따르면 테잎에 기록된 임의의 기계를 그대로 흉내낼 수 있게 된다. 튜링은 이런 능력을 갖춘 기계를 자신이 정의한 기계 부품들로 만들어 보인다.





튜링은 이 궁극의 기계를 *보편만능의 기계*$_{universal\ machine}$라고 부른다. 튜링기계의 하나지만 모든 튜링기계를 흉내낼 수 있기 때문이다. 어떤 계산이건 그에 해당하는 기계를 테잎에 입력으로 받아 그 작동을 그대로 흉내낼 수 있기 때문이다. 그가 정한 범주에 드는 모든 기계적인 계산이 그 하나의 기계로 돌려볼 수 있기 때문이다.

그럼 어떻게 튜링기계의 부품들로 이런 궁극의 기계를 만드는 지 구경가보자.

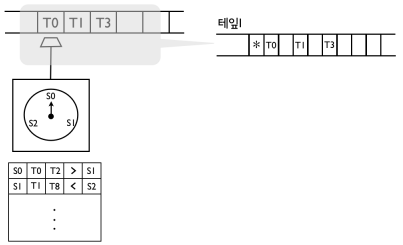
## 튜링기계를 테잎에 표현하기

임의의 튜링기계를 유한 개의 심볼들로 테잎에 표현하는 방법은 간단하다. 일단 세 개의 테잎를 사용해서 각 부품들을 테잎에 표현하는 것으로 하자. 세 개의 테잎은 간단히 하나의 테잎으로 합칠 수 있는데 이 이야기는 잠시 미루자.

튜링기계마다 상태심볼과 테잎심볼들은 각양각색일 것이지만, 상태심볼은 늘 S0, S1, $\cdots$로 바꿀 수 있고, 테잎심볼은 늘 T0, T1, $\cdots$로 바꿀 수 있다. 이런 심볼들을 상태심볼과 테잎심볼들로 사용하는 튜링기계에서부터 시작하자.

- 테잎I: 튜링기계의 테잎과 읽고쓰는 장치 담기.

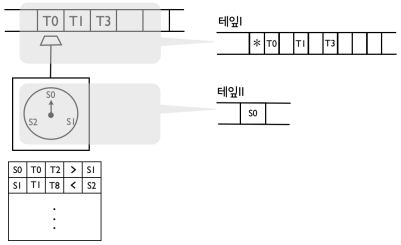
  튜링기계의 테잎은 테잎심볼들을 일렬로 가지고 있는 것이므로, 그 테잎을 그대로 테잎I에 표현하면 된다.



단, 튜링기계가 현재 가르키는 (⏃) 테잎 칸을 테잎I에 표현해 놓아야 한다. 그래야 그 표시를 찾아가서 테잎을 읽고쓰는 일을 흉내낼 수 있다. 이 위치 표시가 $*$이다. 그래서 테잎I 에는 각 칸을 두 칸의 짝으로 구성해서 왼쪽칸은 위치 표시를 놓는데 쓰고 오른쪽칸에는 원래 심볼을 쓴다. 튜링기계가 다음칸으로 이동하는 것은 테잎I에서는 위치 표시가 이동하는 것이다.

- 테잎II: 튜링기계의 현재 상태심볼 담기.

  튜링기계의 현재 상태심볼(S0, S1, $\cdots$ 중 하나)은 테잎II에 써넣고 읽고 쓸 준비를 하면 된다.




- 테잎III: 튜링기계의 규칙표 담기.

  어떤 작동규칙표던지 몇 개 안되는 고정된 심볼들을 이용해서 일렬로 표현할 수 있다. 사용하는 심볼은 S, T, >, <, ||, 0,$\cdots$, 9, X 이다. 심볼 X는 각 규칙의 경계를 표시하는데 쓴다. 예를 들어, 규칙이 두 개인 아래와 같은 작동규칙표는

| | | | | |
|---|---|---|---|---|
| S0 | T0 | T1 | > | S1 |
| S1 | T2 | T12 | < | S0 |

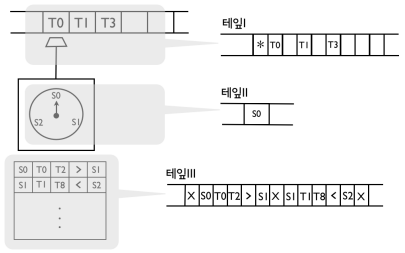
  일렬로 다음과 같이 표현한다.

XS0T0T1>S1XS1T2T12<S0X



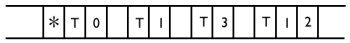


이렇게 표현한 작동규칙표를 테잎III에 담는다.




이렇게 어떤 튜링기계라도 세 개의 테잎(테잎I, 테잎II, 테잎III)에 표현할 수 있고, 사용하는 심볼은 항상, 고정된 17개면 충분하다: S, T, <, >, ||, 0,$\cdots$, 9, X, $*$.

단, 간단히 설명하려고 살짝 지나친 것이 있는데, 위 그림들에서 테잎 한 칸에 쓴 상태심볼과 테잎심볼들은 사실 여러 칸을 차지한다. 뒤에 붙은 숫자때문이다. 예를 들어, 테잎I은 사실상 다음과 같은 모습이다.




## 테잎에 표현한 튜링기계를 돌리는 규칙표

세 개의 테잎에 표현된 튜링기계를 돌리는 작업은 간단하다. 다음의 싸이클을 반복하면 된다.

테잎II에서 현재상태 심볼 S$i$을 읽는다. 테잎I에서 위치심볼($*$)을 찾아 그 곳의 테잎심볼 T$j$을 읽는다. 이렇게 읽은 현재상태 S$i$와 읽은심볼 T$j$와 매치되는 규칙을 테잎III에서 찾는다. 그리고 이렇게 찾은 규칙 | S$i$ | T$j$ | T$j'$ | $m$ | S$i'$ | 이 시키는 일을 한다. 즉, 테잎에 쓸 심볼 T$j'$과 다음 상태 심볼 S$i'$을 테잎I과 테잎II에 옮겨 쓰고, 찾은 규칙에 명시된 움직임 $m$대로 ($m$이 >이면 오른쪽, <이면 왼쪽,

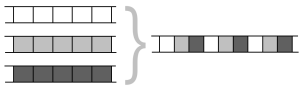
||이면 제자리) 테잎I의 위치심볼(∗)을 옮긴다.

이 반복 작업을 튜링기계의 작동규칙표로 표현할 수 있다. 이 작업들은 튜링기계가 할 수 있는 단순한 것들이기 때문이다. 하는 일의 핵심은 두 가지 뿐이다. 주어진 심볼을 앞부분에 가진 문자열(규칙) 찾기와, 심볼을 복사해서 옮기기다.

이것이 *보편만능 튜링기계*$_{universal\ machien}$다. 임의의 튜링기계가 테잎에 올라오고 가지고 있는 작동규칙표대로 일을하면, 테잎에 올라온 튜링기계의 작동을 그대로 따라하는 것이 된다.

그러데 잠깐, 그런 기계가 튜링기계는 아니지 않은가? 테잎이 하나가 아니고 세 개이지 않은가? 그래서 읽고쓰는 장치도 세 개여야 하고, 규칙표도 세 개의 테잎에서 읽은 심볼과 쓸 심볼 그리고 다음 움직임등을 각각 지정해야 한다. 튜링기계의 작동을 모사하기 위해서 튜링기계의 능력을 벗어난 기계를 만든건 아닌가?

그렇지 않다. 세 개의 테잎을 사용하더라도 하나의 테잎을 사용하는 튜링기계로 볼 수 있다. 테잎 세 개를 테잎 하나로 합칠 수 있기 때문이다. 각 테잎의 칸들을 번갈아서 한 테잎에서 깍지껴서 표현하면 된다.



각 테잎에서 따로 읽고쓰던 건 하나로 합쳐진 테잎에서는 일정 보폭으로 건너뛰면서 읽고쓰면 된다. 각 테잎마다 달랐던 읽는 위치는 합쳐진 테잎에서는 칸 옆에 표시해 놓으면 된다. 그 표시를 위해서 칸마다 둘로 나눠서 왼쪽 칸을 해당 테잎의 위치심볼을 써 넣는 장소로 사용하면 된다.





### 2.1.9 급소

이제 튜링의 증명이 기대는 중요한 사실을 이야기하자. 튜링기계의 한계와 직통으로 연결된 급소 이야기다. 무한한 것들에 존재하는 크기의 차이때문에 생긴 급소다.

의아하겠지만 무한에도 크기의 차이가 있다. 무한하다고 다 같은 무한이 아닌 것이다. 예를 들어서, 자연수(ℕ)의 개수와 실수(ℝ)의 개수가 모두 무한하지만 실수의 개수가 훨씬 더 크다. 또, 자연수의 개수보다 자연수의 부분집합의 개수가 훨씬 더 크다. 칸토르(Cantor)가 처음으로 확인해 준 사실이다. 같은 무한이래도 자연수(ℕ)만큼 있으면 *셀 수 있을만큼 많은*$_{countably\ many}$ 것이지만 실수(ℝ)만큼 많으면 *셀 수 없이 많다*$_{uncountably\ many}$.

이 차이를 구분하는 경계에 튜링기계의 한계가 놓여있다. 튜링기계의 개수는 무한히 많지만 셀 수 있을만큼만 많다. 자연수의 개수를 넘지 못한다. 이 때문에 튜링기계로는 참인 명제를 모두 만들어 낼 수 없게된다.

튜링의 증명은 다음 절에서 구경하도록하고, 여기서는 왜 튜링기계의 개수가 자연수의 개수를 넘지 못하는지 알아보자. 이유는 간단하다. 테잎에 표현된 튜링기계는 고유의 자연수 하나로 표현할 수 있기 때문이다.

위에서 살펴본 바와 같이 튜링기계를 하나의 테잎에 일렬로 표현할 수 있고, 일렬에 들어가는 심볼들은 S,T,<,>,||, 0,⋯,9, X,∗ 등 17개면 충분했다. 이 심볼들 문자열이 튜링기계 하나를 표현하게 된다.

따라서 이 문자열을 고유의 자연수로 변환하는 방법은 많이 있을 수 있다. 한 방법은 이 문자열을 17진수의 수로 해석해서 자연수로 변환하는 것이다. 이 자연수를 다시 17진수로 변환하면 원래의 문자열로 복원될 것이고, 이것을 다시 그림으로 복원하면 원래의 튜링기계 모습으로 되돌아갈 것이다.

예를들어, 위의 심볼들이 차례대로 0(S) 부터 16(∗)을 뜻하는 것으로 정하면, 심볼들의 일렬이 아래와 같다면 17진법으로 표현한 숫자일 뿐이고

```
XS1T2>*
```

상식적인 10진법 자연수로는 (X는 15, S는 0, 1는 6, T는 1, 2는 7, >는 3, ∗는 16 이므로) 아래와 같다.

$$\underset{\texttt{X}}{15\times17^6}\ \underset{\texttt{S}}{+0\times17^5}\ \underset{\texttt{1}}{+6\times17^4}\ \underset{\texttt{T}}{+1\times17^3}\ \underset{\texttt{2}}{+7\times17^2}\ \underset{\texttt{>}}{+3\times17^1}\ \underset{*}{+16\times17^0}$$

즉, 362571664 이다.

그러므로 튜링기계의 개수는 자연수의 개수를 넘지 못한다. 이 세상의 모든 튜링기계마다 고유의 자연수가 지정될 수 있기때문이다.

이 한계가 튜링기계의 능력을 제한하는 급소다. 즉, 현재 컴퓨터의 급소다. 이 한계 때문에 튜링기계는 참인 명제를 모두 만들어낼 수 없다. 어떻게 그렇게 되는지 튜링의 증명을 구경가보자. 다음 절이다.

### 2.1.10 증명

"그 기계(궁극의 기계)를 타고 (괴델의 불완전성) 증명의 중심부 협곡을 아슬아슬 통과해"(2.1.6절) 가는 과정은 이렇다. 증명할 것은 모든 참인 명제를 차례로 만들 수 있는 튜링기계 – $A$라고 하자 – 는 존재하지 않는다는 것이다:

증명목표: $A$는 존재하지 않는다.

궁극의 기계는 임의의 기계를 흉내낼 수 있다고 했다. 그렇다면, 테잎에 표현된 기계를 흉내내는 것 대신에 테잎에 표현된 기계가 멈출지 안멈출지(*멈춤문제*$_{halting\ problem}$)를 정확히 판단하는 기계 $H$는 가능할까?

만일 모든 참인 명제를 차례로 만들 수 있는 튜링기계 $A$가 존재한다면 멈춤문제를 푸는 그런 기계 $H$를 쉽게 만들 수 있다. 다음과 같이 만들면 된다. 멈출지 여부를 판단해야 하는 튜링기계 $M$을 받으면, 궁극의 기계로 튜링기계 $A$를 흉내낸다. $A$는 모든 참인 명제를 차례차례 모두 만들어낼 것이므로, 언젠가는 "튜링기계 $M$은 멈춘다"는 명제를 만들거나 아니면 그 부정인 "튜링기계 $M$은 멈추지않는다"는 명제를 만들것이다. 기다리면 이 둘 중의 한 명제를 만





들어 낼 것이므로 그걸 보고 $M$의 멈춤여부를 판단하면 그만이다. 이렇게 궁극의 기계와 $A$를 이용해서 멈춤문제를 푸는 튜링기계 $H$를 간단히 만들 수 있다.

사실: $A$가 존재하면 $H$가 존재한다.

이어서 튜링의 마지막 펀치가 날아온다. 멈춤문제를 푸는 기계 $H$는 존재할 수 없다는 증명이다. 이 증명을 어떻게 했는지는 다음 절(2.1.11)로 일단 미루자. 아무튼, 임의의 튜링기계를 받아서 멈출지 여부를 정확히 판단하는 튜링기계는 존재할 수 없다.

사실: $H$는 존재하지 않는다.

그러므로, 증명의 마지막 스텝은 명백하다. 모든 참인 명제를 차례로 만들 수 있는 튜링기계 $A$가 있으면 $H$는 만들 수 있는데, $H$는 존재하지 않으므로 $A$도 존재할 수 없는 것이다. 최종 증명이 이렇게 완성된다.

이제 이 증명이 기댔던 핵심 파트 – *멈춤문제*$_{halting\ problem}$를 푸는 튜링기계는 존재할 수 없다 – 를 튜링이 어떻게 증명했는 지 구경가보자. 이 증명에는 튜링기계가 자연수라는 사실과, *보편만능의 기계*$_{universal\ machine}$와 *대각선 논법*$_{diagonalization}$이 동원된다. 이 파트를 끝으로 튜링의 컴퓨터 원조논문 구경을 마무리하자.

### 2.1.11 멈춤문제의 증명

*멈춤문제*$_{halting\ problem}$를 푸는 튜링기계는 존재할 수 없다는 증명의 얼개는 이렇다. 튜링기계는 자연수의 개수 만큼있다. 그런데 멈춤문제를 정확히 풀어주는 튜링기계 $H$가 있다고 하면, 그 $H$를 이용해서 튜링기계의 개수가 자연수보다 많게 된다는 것을 보일 수 있다. 이건 모순이다. 따라서 멈춤문제를 푸는 튜링기계는 존재할 수 없다.

어떻게 $H$가 존재하면 자연수보다 많은 튜링기계가 가능할까? 그 증명은 *대각선 논법*$_{diagonalization}$을 이용한다. 칸토르(Cantor)가 자연수의 개수보다 자연수의 부분집합의 개수가 더 많다는 것을 보일때 사용한 방법이다.

우선, 튜링기계의 개수는 자연수 개수를 넘을 수 없으므로, 모든 튜링기계마다 자연수로 번호 $1, 2, 3, \cdots$를 붙이면 모두 커버할 수 있다. 튜링기계의 테잎에 올라온 입력도 그렇다. 유한 개의 테잎 심볼로 유한한 길이만큼 써 넣은 것이므로 그 가짓수는 자연수의 개수보다 많을 수 없다. 입력마다 번호 $1, 2, 3, \cdots$을 붙이면 모든 입력이 커버된다.

이제, 모든 튜링기계마다 모든 입력에 대해서 $H$의 결과를 테이블로 만들어서 채워넣을 수 있다. 세로축에는 자연수 개수 만큼인 모든 튜링기계들이 도열해 있고, 괄호축에도 자연수 개수만큼인 모든 입력들이 도열해 있다. 각 칸에 해당하는 튜링기계와 입력을 $H$에게 줘서 멈춤문제을 풀도록 한다. 끝난다고하면 1을 써 넣고 안끝난다고 하면 0을 써 넣는다. $H$는 모든 튜링기계에 대해서 그런 답을 정확히 할 수 있다고 했으니, $H$를 이용해서 그렇게 테이블을 채울 수 있는 것은 당연하다.

| | | 입력 | | | |
|---|---|---|---|---|---|
| | | $I_1$ | $I_2$ | $I_3$ | $\cdots$ |
| 튜 | $M_1$ | 1 | 1 | 0 | $\cdots$ |
| 링 | $M_2$ | 1 | 0 | 1 | $\cdots$ |
| 기 | $M_3$ | 1 | 0 | 1 | $\cdots$ |
| 계 | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\cdots$ |

이제 이 테이블을 대각선으로 참조해가면서 모든 튜링기계와 다른 튜링기계를 만들 수 있다. 이 새로운 기계는 다음과 같이 작동한다. 입력 $I_k$에 대해서 테이블을 본다. $M_k$ 행에 뭐가 쓰여있는지. 0이면(즉, $M_k$는 입력 $I_k$를 받으면 안 끊난다) 1을 내 놓고 끝나도록 한다. 1이면(즉, $M_k$는 입력 $I_k$를 받으면 끊난다) $M_k$를 입력 $I_k$에 대해서 돌려서(보편만능의 튜링기계로 흉내내서) 나온 결과에 1을 더해서 내 놓는다.

이렇게 만든 기계는 튜링기계인가? 그렇다. 하는 일이라는 게, 다른 튜링





기계(멈춤문제를 푸는 튜링기계와 보편만능의 튜링기계)를 흉내내고 그에 덧붙여서 기껏 +1을 하는 기계이기 때문이다.

그런데 이런 튜링기계는 모든 튜링기계와 다르다. 1번 튜링기계($M_1$)와는 1번 입력($I_1$)에서 다른 결과를 낸다. 그 입력에 대해서 1번 튜링기계가 안끝나면 1을 내놓고 끝나고, 끝나면 1번 튜링기계의 결과보다 1 큰 결과를 내 놓기 때문이다. 2번 튜링기계와도 같은 이유로 2번 입력에서 갈리고, 등등. 모든 $k = 1, 2, \cdots$에 대해서 $k$번 튜링기계($M_k$)와 $k$번 입력($I_k$)에서 다르게 작동한다. 따라서 이 기계는 모든 튜링기계와 다르다.

이건 모순이다. 모든 튜링기계와 다른 튜링기계가 존재한다니. 자연수 만큼 있는 튜링기계들 모두가 도열해 있는 테이블의 모든 칸을 $H$를 이용해서 메꿀 수 있었다. 그렇게 메꿔진 테이블을 참조해서 모든 튜링기계와 다른 새로운 튜링 기계를 만들 수 있었다. 그런데 이 새로운 튜링기계는 모든 튜링기계가 도열했다는 그 테이블에 이미 있었어야 했다. 그러나 그렇지 않다니 모순이다.

따라서 우리가 참조한 위의 테이블에 문제가 있다. 튜링기계는 자연수만큼 있을 뿐이므로 테이블에 도열한 튜링기계들이 다인것은 확실하다. 그렇다면 문제는 테이블을 메꿔주었던 튜링기계 $H$다. $H$는 아예 존재해서는 안되는 물건이었다.

### 2.1.12 컴퓨터

이제 튜링의 기계부품들을 되돌아보자. 그토록 단순한 부품들로 보편만능의 기계를 만들 수 있다는 것은 놀랍고 반가운 일이다. 튜링의 기계부품은 매우 단순하기 때문에 그 기계부품을 실재 만들어 볼 수 있을게고, 그 부품으로 보편만능의 기계를 만들 수 있지 않겠는가.

이쯤에서 우리 주변의 컴퓨터를 보자. 그곳에는 튜링이 정의한 기계 부품들이 고스란히 구현되어 있다. 테잎은 메모리칩으로, 테잎에 읽고 쓰는 장치는

메모리 입출력 장치로, 작동규칙표는 *중앙처리장치*$_{cpu}$로 구현한 것이다. 따라서 실행시키고 싶은 일에 해당하는 기계를 메모리에 표현해 – 소프트웨어로 만들어 – 넣어주기만 하면 컴퓨터는 그 정의대로 일을 수행한다. 튜링의 보편만능의 기계, 그 실재가 우리 손바닥위에 놓인 컴퓨터인 것이다.

## 2.2 400년

이렇게 튜링이 컴퓨터의 청사진을 우연히 펼치기까지, 논리적인 추론과정이 뭔지를 정의하고 싶어했던 역사는 적어도 400년 정도로 펼쳐진다. 논리 추론이란게 결국 몇가지 방식의 생각 패턴들을 반복해서 사용하는 것 같다는 감을 잡고, 그 생각 패턴을 명확하게 드러내 보고 싶어했던 사람들. 그 생각의 방법을 정확히 표현할 수 있다면, 그것으로 생각을 하인(기계)에게 시킬 수 있으리라고 꿈꾼 사람들. 무르익던 이 꿈이 산산히 조각날 때까지의 400년. 그 동안 그 꿈을 부풀리는 성과와 그 꿈을 깨뜨릴 성과가 동시에 쌓여갔었다.

더 거슬러 올라갈 수 있겠지만 1600년대부터 살펴보면[1] 라이프니쯔(Gottried Libniz, 1646년–1716년)가 초보적이지만 심볼을 사용해서 논리추론의 패턴을 정의하는 것을 시도해봤었다. 200년 후, 프레게(Gottlob Frege, 1848년–1925년)가 체계적인 표기법을 고안해서 많은 논리추론과정을 커버하는 추론의 패턴을 찾아나서기도 했다. 프레게가 고안한 표기법은 현재 기호논리학에서 사용하는 표기법의 원조가 된다. 러셀(Bertrand Russell, 1872년–1970년)과 화이트헤드(Alfred Whitehead, 1861년–1947년)는 프레게의 표기법을 빌려 수학의 모든 논리체계를 엄밀히 규명하려는 대장정에 나서기도 했다. 이런 분위기에서 힐베르트(David Hilbert, 1862년–1943년)가 논리체계를 정확하게 한정해서(*1차 논리*$_{first\text{-}order\ logic}$) 여기서의 추론과정을 모두 담아내는 기계적인 패턴을 찾아나서자는 제안을 하게 된다. 하지만 곧 이어 이 꿈들은 모두 불가능하다고, 괴델(Kurt Gödel, 1906년–1978년)이 증명하고 튜링(Alan Turing, 1921년–1954년)이 재확인한다. 이 증명에는 칸토르(Georg Cantor, 1845년–1918년)가 무한수

[1] 유럽 이야기다. 우리 주변에도 비슷한 시기에 혹은 그 이전에 비슷한 의문을 가지고 궁리한 학자들이 있을법한데, 남아있는 기록을 알 수 없다.





에도 크기 차이가 있다는 사실을 밝히는데 고안한 *대각선 논법*$_{diagonalization}$이 동원된다.

이렇게 3-400년 부풀던 꿈과 철저한 좌절의 과정을 거쳐 컴퓨터의 청사진이 우연찮게 드러나게 된 것이다.

### 2.2.1 의문

아직 풀리지 않는 의문이 있다. 튜링기계를 능가하는 컴퓨터가 가능할까? 즉, 튜링의 방식을 넘어서는 기계적인 계산이란게 있을까?

현재의 컴퓨터가 해내는 기계적인 계산이란 튜링의 정의였을 뿐이다. 절대적인 사실이 아니고, 사실인걸로 접고 넘어가는 컴퓨터의 "파라다임"일 뿐이다.

2036년쯤이 될까? 튜링의 논문이후 100년. 누가 알겠는가. 또 다른 24세의 학도. 어디선가 전혀 관계없을 듯 한 연구에 몰두하는 중에 튜링의 기계를 능가하는 컴퓨터를 암시하는 디자인을 슬쩍 펴보이는 일이 생길지. 희소식은 종종 의외의 곳에서 오지않던가.

*들어보시게, 시절을 뛰어넘어 명창은 한 번 반드시 나타나는 법*

- 송찬호, [임방울]

## 2.3 다른 트랙

한편, 튜링의 논문 이전부터, 계산하는 기계장치는 꾸준히 고안돼왔었다. 고대부터 쓰였던 주판이 그렇고, 파스칼(Pascal)과 라이프니쯔(Libnize)가 1600년대에 만든 사칙연산 계산기와 배비지(Babbage)가 1830년대에 만들고 확장해갔던 자동 계산기가 그렇다. IBM은 1920년대부터 회계장부 숫자들을 자동으로 더하고 빼는 기계를 만들어서 팔고 있었다. 대부분 전기를 쓰지않는, 톱니바퀴등으로 만든 순수 기계장치였다. 그러다가 전기장치를 이용해서 만들어갔다. 통계학자들은 통계계산을 자동으로 하는 기계를 전기장치로 만들었

고(ABC), 더 일반적인 계산이 가능한 자동 계산기를 하버드 대학(Harvard University)이 만들었고(Mark), 펜실베니아 대학(University of Pennsylvania)이 만들었다(ENIAC).

이 트랙의 자동 계산기계들은 튜링의 보편만능의 기계와 가까와지고 있었다. 하나의 기계로 여러 다양한 계산이 가능하도록 만들어지고 있었다. 튜링의 보편만능의 기계 개념 이전부터였고, 동시대에 일어나고 있었던 일이다. 할 일들이 메모리에 프로그램으로 표현되어 올라가는 방식은 아니었지만 계산해야 할 일들이 그 기계의 입력으로 주어지기 시작했다.

이 트랙의 성과들은 튜링의 근본적인 보편만능의 기계 디자인을 만나면서 와해되고 점프한다. 디자인은 튜링의 보편만능의 기계 설계에 접속되면서 그렇게 정리되고, 그동안 쌓였던 구현기술들은 튜링의 보편만능의 기계를 구현하는데 신속하게 동원된다.

튜링과 별개로 발전하던 자동계산 기계장치들의 성과들. 마치 제 몸을 흔들며 바람을 일으켰던 나무들. 그리고 튜링이 투척한 근본적인 컴퓨터의 디자인. 큰 바람은 다른 연원에서부터 불어와 나무들을 흔들었다.

*해가 떠오르리라 해가 떠올라 별들이 그 무덤에 파묻히리라*

- 고은, [이름을 묻으며]