언어 정글

프로그래밍 언어의 두 개의 뿌리

- *튜링기계*$_{turing\ machine}$ (Alan Turing)
  - 상위로 상위로: 기계에 명령하는 언어
  - C, Java, C++, C#, JavaScript, Objective C, Python, PHP, Scala, Swift, Rust 등
- *람다계산법*$_{lambda\ calculus}$ (Alonzo Church)
  - 상위로 상위로: 값을 계산하는 언어
  - ML, OCaml, Haskell, F#, Lisp, Clojure, Rua, Python, Scala, C++14 등
- 최신 언어들은 두 방식 모두를 지원

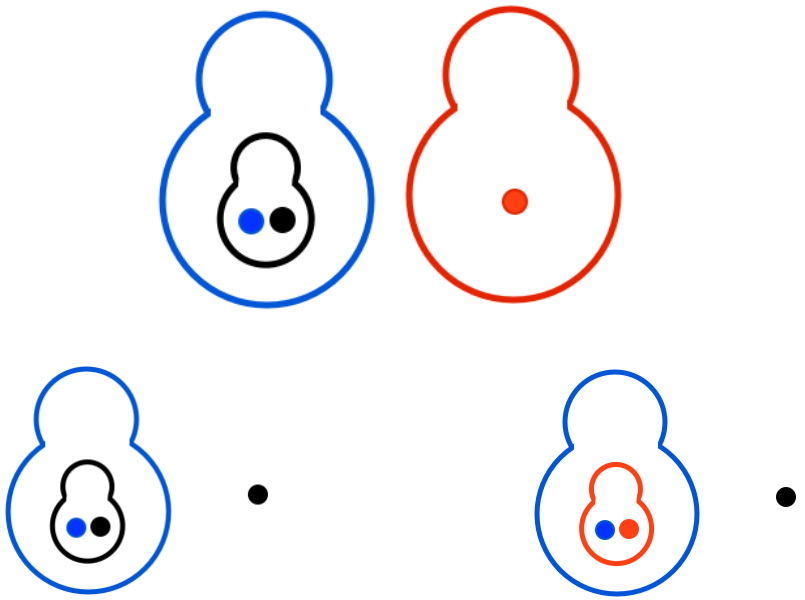
기계의 중력 vs 람다의 중력

- 기계의 중력
  - 명령하며 기계상태를 변화시키는 주문
  - 요리법, 장보기목록

```
a ← 0
b ← read
a ← a+b
b ← b+1
```

- 람다의 중력
  - 기계는 없다. 값을 계산하는 식. "산수 스타일"
  - 초등학교때부터 보아온 산수 언어. $\sum_{n=1}^{10} n^2$
  - "$P \subset Q$ 이고 $P \neq Q$라고하자. 그러면 $P^c \cup Q = U$ 이고 $P \cap Q^c = \emptyset$ 이고 $P \cup Q^c \neq U$이다."
  - 프로그래밍 언어의 요건 $\supseteq$ 수학 언어의 요건
- 다른 기원은 다른 방식의 생각을 유도하는 프로그래밍 언어를 탄생시킴

람다 계산법(lambda calculus)(1/2)



x	구슬(색깔대신 이름 x)
$\lambda x.E$	눈사람(색깔은 x)
$E\ E$	나란히 있기

람다 계산법(lambda calculus) (2/2)

$$
\begin{aligned}
\underline{3} &= \lambda s.(\lambda z.\underbrace{s(s(s}_{3}\,z))) \\
\underline{+}\,n\,m &= \lambda s.(\lambda z.ns(msz)) \\
\underline{\times}\,n\,m &= \lambda s.(\lambda z.n(ms)z) \\
\underline{T} &= \lambda x.(\lambda y.x) \\
\underline{F} &= \lambda x.(\lambda y.y) \\
\underline{and}\,a\,b &= a\,b\,\underline{F} \\
\underline{\mathsf{zero?}}\,n &= n\,(\lambda x.\underline{F})\,\underline{T} \\
\underline{\mathsf{repeat}}\,E &= (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))\,E
\end{aligned}
$$

논리는 언어의 거울

(람다 중력권)

컴퓨터 세계에서 언어와 논리는 동전의 양면일 뿐, 같은 것이다

증명하기	$\longleftrightarrow$	프로그램짜기
"논리적인 비약없이 새로운 사실을 확인해가는 과정."	$\longleftrightarrow$	공짜없이 새로운 데이터를 만들어가는 과정.
"참인 사실 혹은 사실이라고 가정한 것들로부터 시작."	$\longleftrightarrow$	기초적인 데이터에서 부터 시작. 기초적인 데이터는 정수, 문자, 참거짓 등.
"사실을 기반으로 해서 새로운 사실들을 만듬."	$\longleftrightarrow$	이미 만든 데이터를 가지고 새로운 데이터들을 만듬.
"만들어가는 과정은 근거없는 건너뛰기 없이, 논리적으로 누구나 수긍하는 추론의 징검다리를 밟고 가는 과정만 있다."	$\longleftrightarrow$	새 데이터를 만드는 과정은 공짜가 없다. 사용하는 프로그래밍 언어에서 제공하는 프로그램 조립방식을 써서 만든다.

논리 추론의 징검다리

$$\frac{A \quad B}{A \wedge B} \qquad \frac{A \wedge B}{A} \qquad \frac{A \wedge B}{B}$$

$$\frac{\begin{array}{c}\overline{A} \\ \vdots \\ B\end{array}}{A \Rightarrow B} \qquad \frac{A \Rightarrow B \quad A}{B}$$

$$\frac{A}{A \vee B} \qquad \frac{A}{B \vee A} \qquad \frac{A \vee B \quad \begin{array}{c}\overline{A} \\ \vdots \\ C\end{array} \quad \begin{array}{c}\overline{B} \\ \vdots \\ C\end{array}}{C}$$

논리 증명나무

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(A \Rightarrow B) \wedge (A \Rightarrow C)}{A \Rightarrow B} \quad A}{B} \quad \dfrac{\dfrac{(A \Rightarrow B) \wedge (A \Rightarrow C)}{A \Rightarrow C} \quad A}{C}}{B \wedge C}}{A \Rightarrow (B \wedge C)}}{(A \Rightarrow B) \wedge (A \Rightarrow C) \Rightarrow (A \Rightarrow (B \wedge C))}$$

거울: 증명하기 = 프로그램짜기 (1/2)

$$\frac{A \quad B}{A \wedge B} \quad \longleftrightarrow \quad \text{데이터 뭉치 만들기}$$

$$\frac{A \wedge B}{A} \quad \frac{A \wedge B}{B} \quad \longleftrightarrow \quad \text{데이터 뭉치 사용하기}$$

$$\frac{\begin{array}{c}\overline{A} \\ \vdots \\ B\end{array}}{A \Rightarrow B} \quad \longleftrightarrow \quad \text{함수 만들기}$$

$$\frac{A \Rightarrow B \quad A}{B} \quad \longleftrightarrow \quad \text{함수 사용하기}$$

거울: 증명하기 = 프로그램짜기 (2/2)

$$\frac{A}{A \vee B} \quad \longleftrightarrow \quad \text{데이터 뭉뚱그리기}$$

$$\frac{A \vee B \qquad \begin{array}{c}\overline{A}\\ \vdots \\ C\end{array} \qquad \begin{array}{c}\overline{B}\\ \vdots \\ C\end{array}}{C} \quad \longleftrightarrow \quad \text{뭉뚱그린 데이터 사용하기}$$

$$\overline{A} \quad \longleftrightarrow \quad \text{함수의 인자를 사용하기}$$

거울의 효능

- 프로그램 짜기전: 짤 프로그램의 구도잡기
  - *타입*$_{type}$이라는 언어로 겉 얼개를 표현
  - 참 논리식 $\longleftrightarrow$ 타입
- 프로그램 짜고나서: 짠 프로그램이 무난히 작동할지 살피기
  - 무난히 작동 $=$ 타입에 맞게 돈다
  - 논리분야의 타입이 활용되던 모습(1950-1970년대)
  - 그렇다면 프로그램에서도 가능하겠지(1970-현재)

짤 프로그램의 구도잡기

타입으로("보통명사"로) 프로그램 얼개를 표현

$$
\begin{aligned}
\text{결혼하기} &: \text{남자} \times \text{여자} \to \text{부부} \\
\text{사랑하기} &: \text{부부} \to \text{사랑} \\
\text{아기만들기} &: \text{부부} \times \text{사랑} \to \text{아기} \\
\text{가족만들기} &: \text{남자} \times \text{여자} \to \text{가족}
\end{aligned}
$$

```
가족만들기(x,y) =
    let  couple = 결혼하기(x,y)          ('남자'와 '여자'가 '부부'를 만들고)
    let  love = 사랑하기(couple)          ('부부'가 '사랑'을 만들고)
    let  baby = 아기만들기(couple, love)  ('부부'가 '사랑'으로 '아기'를 만들고)
     in  (couple, baby)                   (그런 '부부'와 '아기'가 '가족'이다)
```

짠 프로그램이 무난히 작동할지 검진하기 (1/2)

프로그래머의 불안

- 기획한 얼개에 따라 잘 메꾼걸까?
- 그래서 제대로 작동하는 프로그램일까?
- 만 페이지가 넘는, 대하소설 보다 복잡한 흐름
- 프로그램 실수는 고스란히 드러날 것이다

프로그램 검진 vs 다른 분야

만든것	프로그램	$\longleftrightarrow$	기계/건물/약물 디자인
실행기	컴퓨터	$\longleftrightarrow$	자연
바램	실행에대해 미리 확인	$\longleftrightarrow$	작동에대해 미리 확인
안심	“생각대로 돌것이다”	$\longleftrightarrow$	“생각대로 작동할것이다’
확인도구	컴퓨터과학의 성과	$\longleftrightarrow$	자연과학의 성과

짠 프로그램이 무난히 작동할지 검진하기 (2/2)

자동검진이 가능

- 과거: 개인의 기예에 의존하던 프로그램의 완성도
- 현재, 미래: 누구나의 기술로 한 발 전진
  - 자동 검진
- 컴퓨터는 더 이상 맹목적이지 않다
- 프로그램을 검진해서 통과한 것만 실행

$f(x) = x+1$ $\quad$ *sum*$(f) = f(0)+f(1)$ $\quad$ *무게(춘향이)+무게(그네)*

요약(abstraction)의 그물

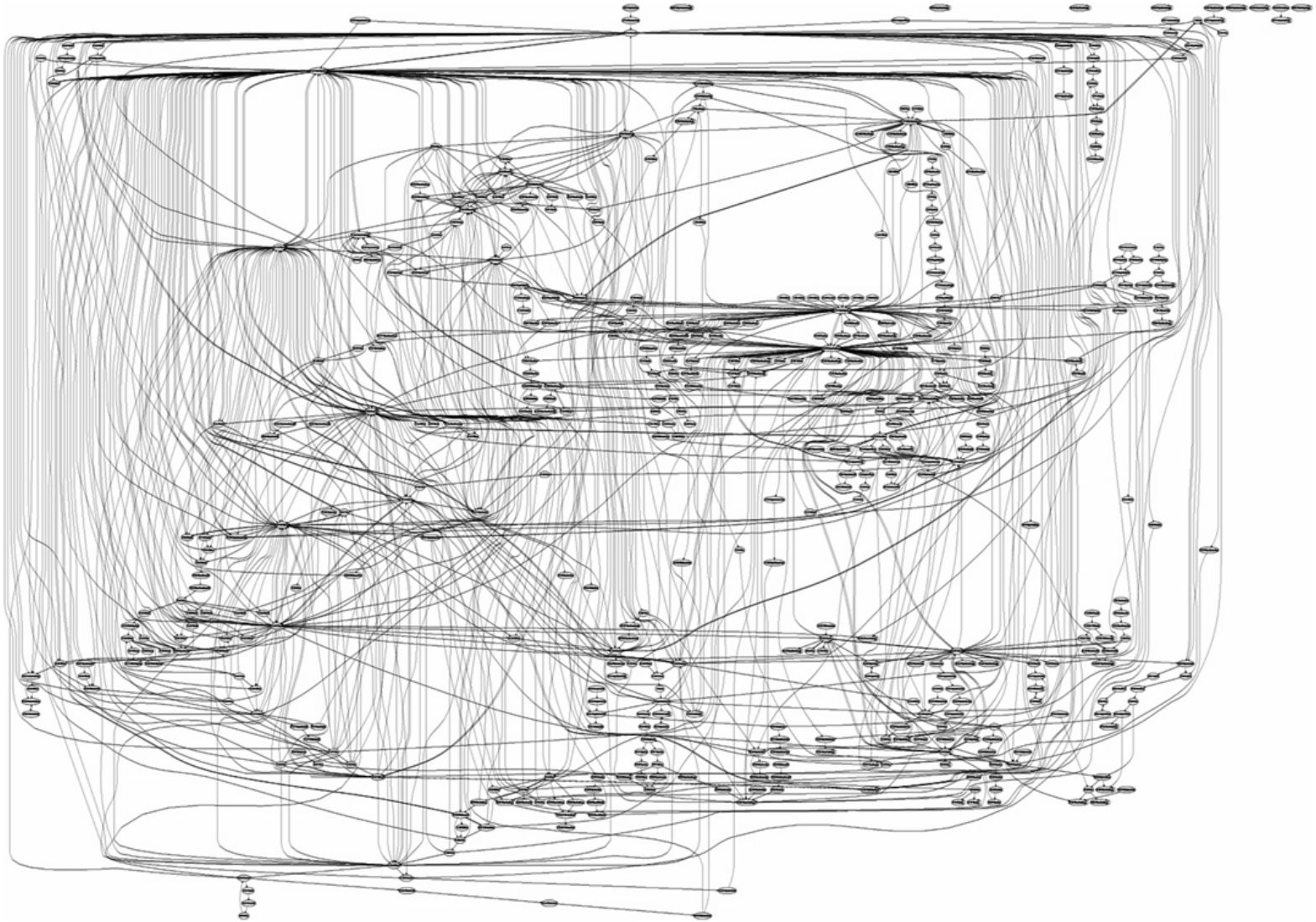
컴퓨터과학에서 늘 동원하는 지혜: "모두 포섭하면서 간단히"

- 프로그램 검진에서도: 요약
  - 모든 디테일을 보는 것은 비현실적
  - 외부입력 가짓수가 너무 많고, 만들어지는 현상이 너무 복잡: *기하급수로 커지며 속수무책*$_{\textit{combinatorial explosion}}$
- 다양한 수준의 요약을 동원:

$$10 \times \textit{무게}(\textit{춘향이}) + 8$$

  - 타입으로 요약: 정수
  - 양수음수로 요약: 양수
  - 구간으로 요약: 100과 3000사이

SW는 복잡하다



SW는 복잡하다

less-382(23,822 LoC)

자연물 만큼 복잡

포유류 뇌의 1만개 뉴런의 넷트웍, Ecole Polytechnique Fédérale de Lausanne, *Blue Brain Project*, 2008

SW 제작의 어려움

- 프로그램의 규모와 복잡도가 점점커짐
  - 프로그램 복잡성의 증가속도 $>>$ hw 성능의 성장속도
  - "sw는 가스다." "Software is gas."
- 프로그램은 기계가 자동으로 실행함
  - 기계는 우리가 바라는 바를 실행하지 않음
  - 기계는 sw에 적힌 바를 실행할 뿐

    "장보기 = 우유 1리터, 신라면 4봉지, 그리고 쌀과자두 사오기."

  - 모든 상황을 고려해야 $\times$ 사소한 실수가 없어야
- 프로그램의 실행을 "미리 완벽히 알기"가 어려움
  - 자동으로는 불가능: 증명됨(1936년, Alan Turing)
  - 사람이 확인해가야: 다양한 도구로 비용절감
  - 현재의 기술수준: 걸음마 "3발짝", 초보수준

고백: 컴퓨터분야는 아직 미숙합니다

분야의 역사 $\sim$ 겨우 5-60년

그래서

- 미숙하기 때문에 $\Longrightarrow$ 기회가 많다.
- 미숙하기 때문에 $\Longrightarrow$ 지금 "Newton", "Galileo", "Curie", 와 같이 살고있다.

컴퓨터분야 발전 속도가 빠르다? (1/3)

- 컴퓨터속도: 10^7배/80년 발전

10^3flops/cpu ENIAC(1945)

$\longrightarrow$ (2×10^{10}flops/core IBM Sequoia(2010))

$\longrightarrow$ 6×10^{10}flops/core Fujitsu Fugaku(2020)

컴퓨터분야 발전 속도가 빠르다? (2/3)

다른 분야와 얼추 비슷: $10^7 \sim 10^9$배 발전/100년

- 에너지분야: 10^7배/100년 발전

  10^{-1}hp/hr (하인) $\longrightarrow$ 1.35×10^6 hp/hr (원전 1GW/hr)

- 교통분야: 10^8배/100년 발전

  10^3j (가마) $\longrightarrow$ 10^{11}j (Delta II), 10^{10}j (누리호)



컴퓨터분야 발전 속도가 빠르다? (3/3)

- 하드웨어: 다른 분야와 비슷한 발전
- 소프트웨어: 많이 미개함
  - 크기만 증가:

    휴대폰 100만줄

    아래아한글 200만줄

    스마트폰 1000만줄

    윈도우 3000만줄
- 소프트웨어: 다른 분야가 이루어 놓은 것을 아직 이루지 못했슴

무엇일까?

다른 분야의 현재 수준

- 제대로 작동할 지를 미리 검증할 수 없는 기계/회로/공정/건축 설계는 없다.
- 제대로 작동할 지를 미리 검증할 수 없는 백신 설계는 없다(?)
- 제대로 작동할 지를 미리 검증할 수 없는 금융상품 설계는 없다(?)

뉴튼방정식, 미적분 방정식, 통계역학, 네비에-스톡스 방정식...

모든 기술의 질문

우리가 만든 것이
우리가 의도한대로 움직인다는 것을
어떻게 미리
확인할 수 있는가?

(사실, 일상에서도 잦은 질문과 답: 입학시험, 입사시험, 궁합, 클럽관리)

소프트웨어 분야에서의 이 질문

작성한 SW의 오류를
자동으로 미리 모두 찾아주거나,
없으면 없다고 확인해주는
기술들은 있는가?

그래서, SW의 오류때문에 발생하는
개인/기업/국가/사회적 비용을
절감시켜주는 기술들은 있는가?

프로그램 검진: 타입오류를 넘어서

SW 오류(bug)

- 소프트웨어에 있는 오류
- 소프트웨어가 생각대로 실행되지 않는 것
- 사람이 소프트웨어를 잘못 만들었기 때문
  - 천재지변이 아님

SW 오류의 예

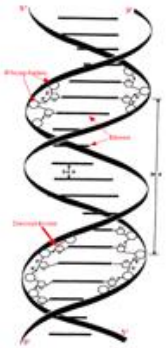
어머니가 전해준 슈퍼마켓 심부름 목록(프로그램)

1. 우유 1리터 2병
2. 우유가 없으면 오렌지쥬스 1리터 3병
3. 우유가 있으면 식빵 500그람 1봉
4. 쌀과 자두

만약에: 우유 1리터 1병만 있으면? 쌀과 자두? 쌀과자두?

SW 오류의 문제

DNA 오류



SW 오류

⇓ ⇓

살다가 병이든다 — SW로 동작하는 제품이 고장난다

사회적 비용 — 사회적 비용

해결책 = 경제적기회 — 해결책 = 경제적기회

SW 오류: 사람이 손으로 손쉽게 잡을 수 없다

- 엄청나게 커지고 복잡해 진 소프트웨어
- 새로운 컴퓨팅 환경: 지구 $=$ 컴퓨터
  - 전지구적 네트웍을 타고
  - 불특정 다수의 코드가 내게로 온다(스마트폰 게임/apps)

자동 기술이 필요

SW 오류 검증 기술: 1세대

문법 검증기: 70년대에 완성된 기술. lexical analyzer & parser

오류 = 소프트웨어의 생김새가 틀린 것

1. 우유 1리터 2병
2. 우유가 없으면 오렌지ㅈㅅ 1리터 3병
3. 있으면 빵식 우유가 500그람 1봉
4. 쌀과 자두

SW 오류 검증 기술: 2세대

타입 검증(type checking): 90년대에 완성.
(30년간 프로그래밍언어 분야의 대표 성과)

오류 = 생긴것은 멀쩡한데, 잘못된 값이 계산에 흘러드는 경우

1. 우유를 담은 얼음을 가스 불위에 올려놓고 데운다
2. 식빵을 유리접시에 담고 방아로 빻는다
3. 2의 접시에 1의 우유를 튀긴다
4. 남자와 소나무를 결혼시킨다

SW 오류 검증 기술: 3세대

프로그램분석검증: 아직 미완성

(static analysis, verification, model checking, ...)

오류 = 생긴것도 멀쩡하고 타입도 맞지만, 바라는 바가 아닌 것

오류 = 필요한 조건을 만족시킬 수 없는 경우

1. 우유를 담은 냄비를 가스 불위에 데운다
2. 식빵을 스텐접시에 담고 방아로 빻는다
3. 남자1명과 여자1명을 결혼 시킨다

- 가스불이 포항제철 용광로가 된다면?
- 방아가 현대중공업의 프레스가 된다면?
- 남녀 나이의 차가 100살이된다면?

데이터의 중력: 새 프로그래밍 중력 (1/7)

많아진 데이터, 싸진 컴퓨터

기계 학습$_{machine\ learning}$ *인덕*$_{induction}$하는 프로그램짜기

확률 추론 프로그래밍$_{probabilistic\ pgm'}$ *앱덕*$_{abduction}$하는 프로그램짜기

데이터의 중력: 새 프로그래밍 중력 (2/7)

- *기계 학습*$_{\textit{machine learning}}$ (*인덕*$_{\textit{induction}}$)
  - 입력: 관찰한 데이터 $\{(a_0, b_0), \cdots, (a_k, b_k)\}$
  - 출력: 짐작하는 함수
  - 예: $\{(-1, 0), (2, 3)\}$에서?
- *확률 추론*$_{\textit{probabilistic pgm'ng}}$ (*앱덕*$_{\textit{abduction}}$)
  - 입력: 인과관계 $A \implies B$, 관찰한 데이터 B
  - 출력: 짐작하는 원인 A
  - 예: 시험점수에서? 선호영화에서?
- 얼추짐작을 자동으로/과학적으로
- 주의: 엉망인 데이터, 빠뜨린 인과관계, 관찰못한 데이터

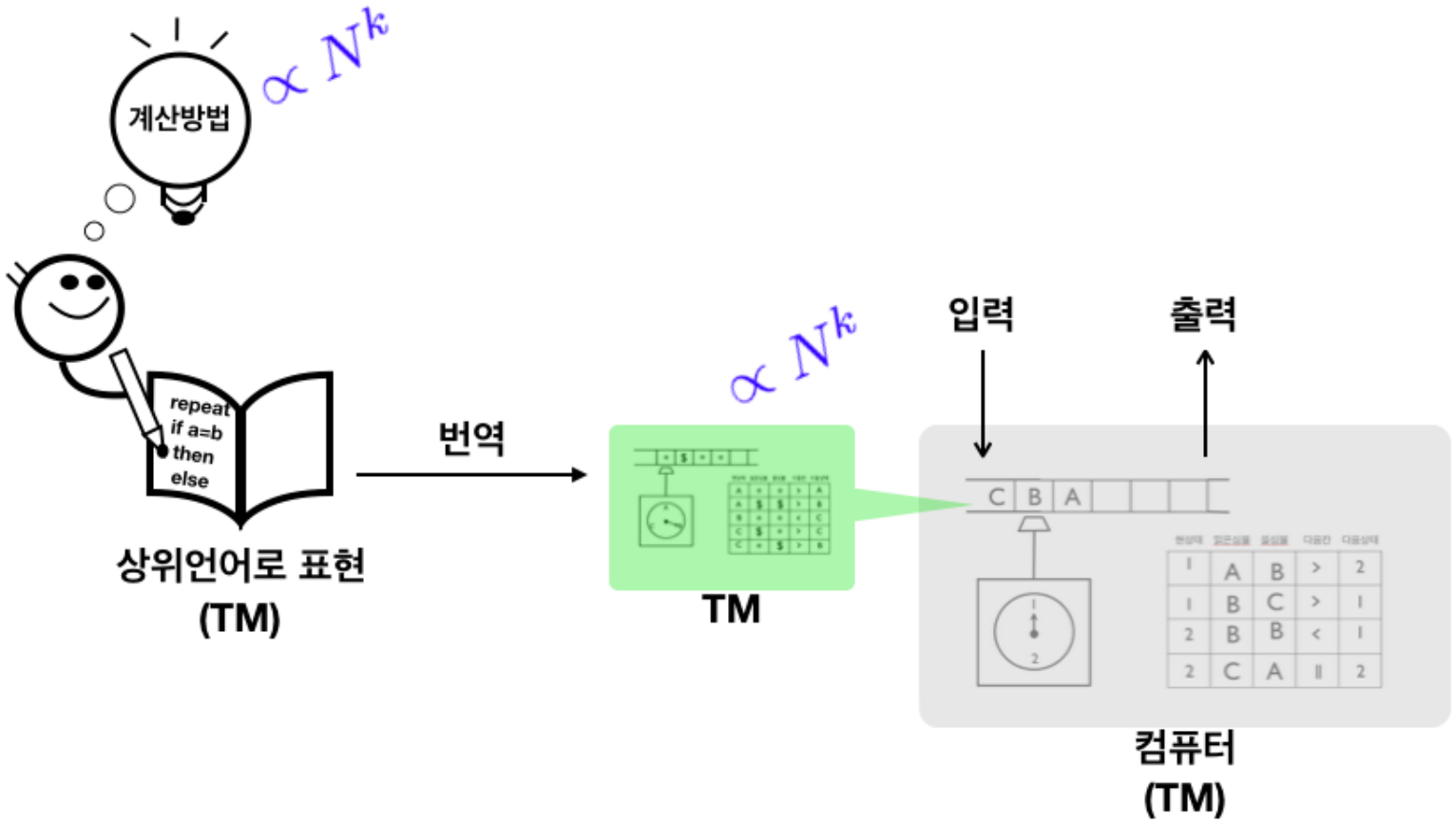
데이터의 중력: 지금까지 프로그래밍 기둥 (3/7)

기둥 A: 기계학습 이전

- 언어로 논리적인 문제풀이법을 표현
- 계산방법을 꼼꼼히 상위의 언어로 손수작성
  - 현실적인 비용의 계산과정이어야
  - 비용 분석, 비용한계 검진 (과학)
- 쓴 글을 "튜링기계"(기계어)로 번역해서 컴퓨터에 싣는다

프로그래밍$_{programming}$, *알고리즘*$_{algorithm}$

데이터의 중력: 지금까지 프로그래밍 기둥 A (4/7)



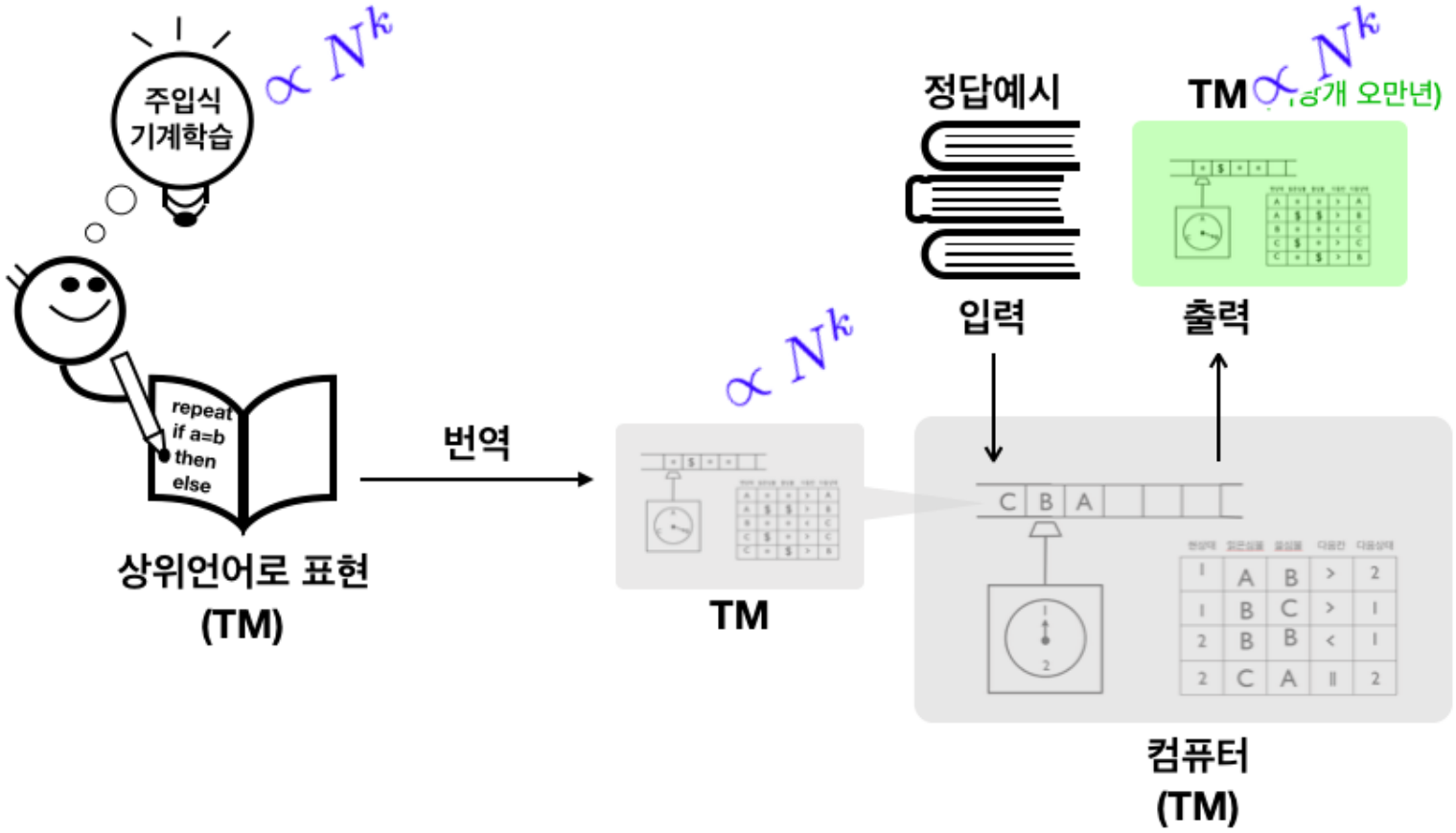
데이터의 중력: 새 프로그래밍 기둥 (5/7)

기둥 Z: 기계학습의 출현

- 어떻하지? 언어로 논리적으로 표현못하겠다
- 정답예시를 주입식으로
  - "보고 따라하길"
- 기계학습: "서당개 5만년" 훈련
  - 수많은 정답예시 주입해서 SW 자동생성
  - 기계학습 과정과 결과 SW 모두
  - 현실적인 비용의 계산과정이어야
  - 학습가능 범주분석, 학습비용 한계 검진 (과학)
- 결과 SW -*뉴럴넷*$_{neural\ net}$- 을 컴퓨터에 싣는다

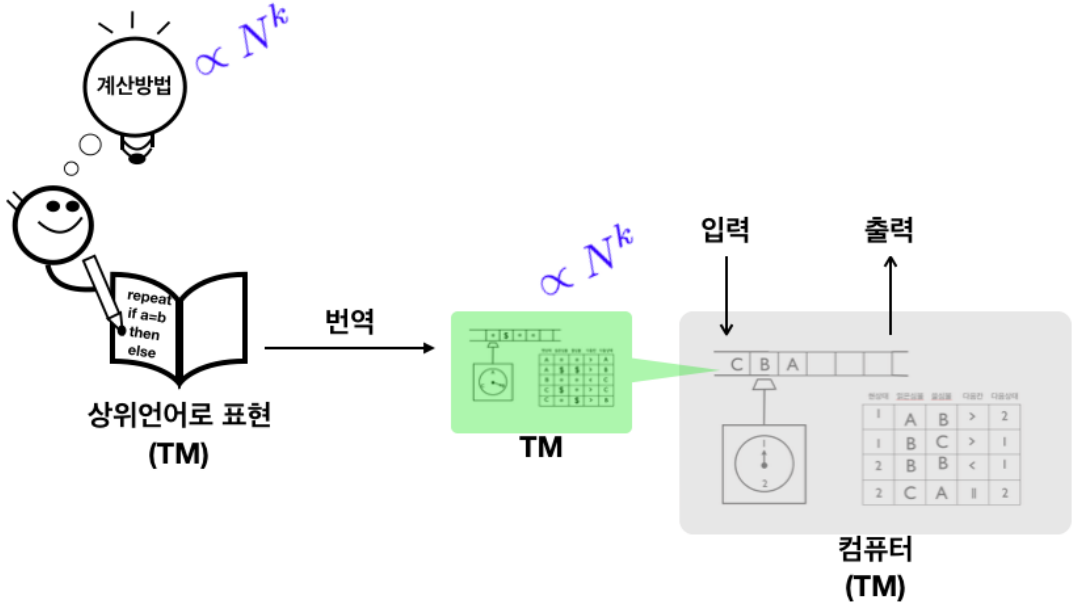
기계학습$_{machine\ learning}$, *에코리즘*$_{ecorithm}$

데이터의 중력: 새 프로그래밍 기둥 Z (6/7)

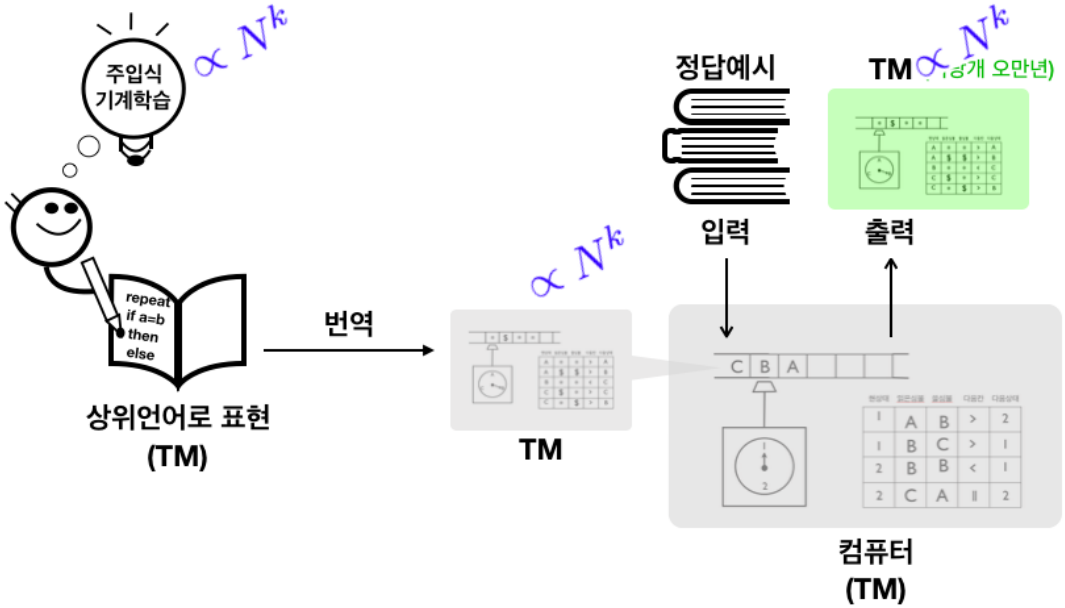


데이터의 중력: 소프트웨어 만들기 두 기둥 (7/7)

기둥 A



기둥 Z



다음