CS422 Chapter #1

마틴 지글러 *

September 7th, 2021

1 Un-/Semi-Decidability and Enumerability

1.1 Undecideable Halting Problem

그 어떤 알고리즘 B도 다음 질문에 항상 정답을 낼 수 없다.

Given $\langle A, \underline{x} \rangle$, does algorithm A terminate on input $\underline{x}$?

Proof. Consider an algorithm B' that on input A, executes B on $\langle A, A \rangle$ and, upon a positive answer, loops infinitely.
How does B' behave on B'? If B' terminates, B' does note terminate. (Contradiction) $\square$

1.2 Un-/Semi-Decidability and Enumerability

Definition (Compute)
*An algorithm $\mathcal{A}$ **computes** a partial function $f :\subseteq \mathbb{N} \to \mathbb{N}$ if it*

- *on inputs $\underline{x} \in dom(f)$ prints $f(\underline{x})$ and terminates,*
- *on inputs $\underline{x} \notin dom(f)$ does not terminate.*

Definition (Decide)
*$\mathcal{A}$ **decides** set $L \subseteq \mathbb{N}$ if it computes its total characteristic function: $cf_L(\underline{x}) := 1$ for $\underline{x} \in L$, $cf_L(\underline{x}) := 0$ for $\underline{x} \notin L$.*

*동한몽

Definition (Semi-decide)

$\mathcal{A}$ ***semi-decides*** *set* $L \subseteq \mathbb{N}$ *if it terminates* <u>*precisely*</u> *on* $\underline{x} \in L$.

Definition (Enumerate)

$\mathcal{A}$ ***enumerates*** L *if it computes some total injective* $f: \mathbb{N} \to \mathbb{N}$ *with* $L = range(f)$

Theorem 1

Every finite L *is decidable.*

Proof. 알고리즘 안에 L 을 하드코딩하면 가능하다. □

Theorem 2

L *is decidable iff its complement* $\bar{L}$ *is.*

Proof. 0과 1을 바꿔서 print하면 된다. □

Theorem 3

L *is decidable iff both* L *and* $\bar{L}$ *are semi-decidable.*

Proof. $\rightarrow$: 자명하다.
$\leftarrow$: 두 알고리즘을 번갈아가면서 한 단계씩 진행시킨다. (Dovetailing) □

Theorem 4

L *is enumerable iff infinite and semi-decidable.*

Proof. $\rightarrow$: infinite 한 것은 자명하다. 그리고 주어진 x 에 대해서 $f(n) = x$ 가 될 때 까지 $n = 0$ 부터 대입한다.
$\leftarrow$: L 을 semi-decide 하는 $\mathcal{A}$ 를 골라서 적당한 m 을 골라 m 단계 진행 후까지 output이 없으면 L 의 임의의 원소를 출력한다. 이 m 은 충분히 커서 $\mathcal{A}$ 가 terminate 될 예정일 경우 문제없이 terminate 할 수 있어야 한다.
하지만 이 상태로는 함수가 injective 하지 않게 되는데, 새로운 알고리즘 $\mathcal{C}$ 를 만들어서 injective 하지 않은 output 을 받아 최종적인 output이 injective 할 수 있게 만든다. 예를 들면 한번 나온 output 을 list 에 저장하는 방식이 있다. □

2 Reduction, Degrees of Undecidability

2.1 Comparing Decision Problems

The Halting problem (considered as subset of $\mathbb{N}$, is semi-decidable, not decidable.)

$$H = \{\langle \mathcal{A}, \underline{x} \rangle : \mathcal{A}(\underline{x}) \text{ terminates.}\}$$

Nontriviality

$$N = \{\langle \mathcal{A} \rangle : \exists y \ \mathcal{A}(\underline{y}) \text{ terminates.}\}$$

Totality problem

$$T = \{\langle \mathcal{A} \rangle : \forall z \ \mathcal{A}(\underline{z}) \text{ terminates.}\}$$

2.2 Reduction

Definition ($\preccurlyeq$)

For $L, L' \subseteq \mathbb{N}$, write $L \preccurlyeq L'$ if there is a computable $f : \mathbb{N} \to \mathbb{N}$ such that $\forall \underline{x} : \underline{x} \in L \Leftrightarrow f(\underline{x}) \in L'$.

Theorem 5

L' semi-/decidable $\Rightarrow$ so L.

Proof. $\underline{x}$ 가 L 안에 있는지 확인하기 위해 $f(\underline{x})$ 가 L' 에 있는지 확인하면 된다. □

Theorem 6

$L \preccurlyeq L' \preccurlyeq L'' \Rightarrow L \preccurlyeq L''$

Proof. 자명하다. □

Corollary 1

$H \preccurlyeq N$ (H is reducible to N, H is at most as difficult as N.)

알고리즘에 input 을 하드코딩하면 무조건 terminate 하므로 가능. 따라서 N 도 undecidable 하다.

Corollary 2

$H \preccurlyeq T$

알고리즘에 input 을 하드코딩하면 무조건 terminate 하므로 가능. 따라서 T 도 undecidable 하다.

Corollary 3
$N \preccurlyeq H$

주어진 알고리즘이 terminate 하는 input을 찾아서 가능.

Corollary 4
$H \not\preccurlyeq \bar{H}$

H 는 semi-decidable 함과 동시에 non-decidable 하므로 위가 성립하면 모순이다.

Corollary 5
$\bar{H} \preccurlyeq T \Rightarrow T \not\preccurlyeq H$

Proof. Given $\langle \mathcal{A}, x \rangle$, convert to $\mathcal{B}$ which on input m simulates $\mathcal{A}(x)$ for m steps and terminates if $\mathcal{A}(x)$ does <u>not</u> terminate within m steps, otherwize deliberately freezes. 아마 여기서도 충분히 큰 m 이어야 할 것이다. □

3 Busy Beaver a.k.a. Rado Function

$$\hat{H} := \{\langle \mathcal{A} \rangle : \mathcal{A} \text{ terminates on input } \epsilon\} \equiv H$$

For algorithm $\mathcal{A}$, let $t(\langle \mathcal{A} \rangle) :=$ # of steps $\mathcal{A}$ makes before terminating, $:= \infty$ if doesn't.

$$\beta(n) := \max\{t(m) : m \leq n,\ t(m) < \infty\} : \text{Busy Beaver function}$$

모든 알고리즘은 어떤 자연수로 인코딩 될 수 있기 때문에 자연수를 argument 로 받는다. 이 함수는 non-decreasing 이다.

Theorem 7
Every computable $f : \mathbb{N} \to \mathbb{N}$ satisfies $f(m) < \beta(m)$ for some (infinitely many) $m \in \mathbb{N}$.

Proof. The following algorithm answers "$\langle \mathcal{A} \rangle \in \hat{H}$?".
whenever $f(\langle \mathcal{A} \rangle) \geq \beta(\langle \mathcal{A} \rangle)$
Compute $N := f(\langle \mathcal{A} \rangle)$. Simulate $\mathcal{A}$ on input ϵ for N steps. If not yet terminated, answer "no". □

4 Ackermann's function

$$A(0, n) = n + 2 \qquad A(l + 1, 0) = 1$$

$$A(l + 1, n + 1) = A(l, A(l + 1, n))$$

- $A(1, n) = 2n + 1$
- $A(2, n) =$ exponential.
- $A(3, n) =$ tetration.
- Diagonal Ackermann: $n \to A(n, n)$, computable, extremely fast growing but slower than beaver function..

5 LOOP programs

Syntax in Backus-Naur form:

$$\begin{aligned} P := {} & x_j := 0 \\ & \mid x_j := x_i + 1 \\ & \mid P; P \\ & \mid \text{LOOP } x_j \text{ DO } P \text{ END} \end{aligned}$$

Semantics:

- $x_1, \dots x_k$ contain input $\in \mathbb{N}^k$
- LOOP executed x_j times.
- Body must not change x_j in LOOP.

이렇게만 하면 LOOP program 은 무조건 terminate 하게 되므로 FREEZE instruction 을 추가하자.

Theorem 8
To every LOOP program $P = P(x_1, \dots x_k$ there exists some $l = l(P) \in \mathbb{N}$ s.t. P on input $\underline{x} \in \mathbb{N}^k$ makes $\leq A(l, n) < \infty$ steps, where $n := \max\left(2, \sum_j |x_j|\right)$.

이 theorem 의 증명은 넘어간다.

diagonal Ackerman's function 은 loop program 으로 표현이 불가능하다. 유한한 루프르 사용해야 하기 때문에 그런듯. 이 프로그램은 큰 숫자를 만들 때에도 하나씩 더해서 만든다. $A(n, n)$ 을 계산할 때에도 그만큼 더하는 짓을 해야 한다.

5.1 Power of LOOP Programs

Definition

Recall bijective $\mathbb{N}^2 \ni (x, y) \to \langle x, y\rangle \in \mathbb{N}$ *and write* $\langle x, y, z\rangle := \langle\langle x, y\rangle, z\rangle$*, etc.*

Lemma

There exists a LOOP program that, given $x, y \in .\mathbb{N}$*, returns* $\langle x, y\rangle \in \mathbb{N}$*.*

Lemma

There exists a LOOP program that, given $\langle x, y\rangle \in \mathbb{N}$*, returns* x*; another one returns* $y \in \mathbb{N}$*.*

Lemma

There exists a LOOP program that, given integers $n \leq \mathbb{N}$ *and* $\langle x_1, \ldots, x_n, \ldots, x_N\rangle$*, returns* x_n*.*

Lemma

There exists a LOOP program that, given $n \leq \mathbb{N}$*,* y *and* $\langle x_1, \ldots, x_n, \ldots, x_N\rangle$*, returns* $\langle x_1, \ldots, y, \ldots, x_N\rangle$*,*