Today we shall look at the undecidability of First-Order logic. The textbook proves that FO-VALID is undecidable by reducing PCP (Post's Correspondence Problem) to FO-VALID. But in today's class, we introduce and explain the notions of decidability and undecidability.

Definition 12.1 [decision problem, characteristic function] A *decision problem*, $S \subseteq \{0,1\}^*$, is a computational problem whose input can be encoded as a binary string and whose output is either yes or no. We can identify the decision problem, S, with its characteristic function:

$$\chi_S(w) \quad = \quad \begin{cases} 1 & \text{if} \quad w \in S \\ 0 & \text{if} \quad w \notin S \end{cases}$$

Types of inputs: There is a natural and efficiently computable 1:1 and onto map between $\{0,1\}^\star$ and the set of natural numbers, **N**. Thus, we may think of our decision problems S as subsets of **N** instead of as binary strings, if we so choose. Similarly, it is easy to encode pairs of binary strings as binary strings, so we may think of our inputs as pairs of binary strings, $S \subseteq \{0,1\}^\star \times \{0,1\}^\star$, or as pairs of natural numbers, $S \subseteq \mathbf{N} \times \mathbf{N}^\star$, etc. □

Example 12.2 Primality testing, PRIME $= \{2, 3, 5, \ldots\}$, is a polynomial-time decision problem, i.e., PRIME $\in$ P. □

Remark 12.3 Every problem in Computer Science can be thought of as a set of decision problems: on input w, produce output $b_1(w)\, b_2(w) \cdots b_\ell(w)$. The difficulty of the problem can be understood in terms of the complexity of the hardest of these decision problems. Thus, in complexity and computability, problems are decision problems and complexity classes are sets of decision problems. □

Definition 12.4 [decidability] S is *decidable* iff there is a program (Turing machine) that computes its characteristic function, χ_S. Other words equivalent to decidable are *computable*, *solvable* and *recursive*. By the definition of χ_S, this program must always halt and output either 0 or 1. A problem is *undeciable* iff it is not decidable. □

History: The history of undecidability starts with Hilbert's challenge in 1901 to show that all of math is decidable. In a few decades after that, several mathematicians around the world came up with independent definitions of what an algorithm or "effective procedure" is:

- Church: lambda calculus,
- Gödel: recursive functions,
- Kleene: formal systems,
- Markov: Markov algorithms (a rewrite system),
- Turing: Turing machines, and
- Post: Post machines

All of these were proved to have the exact same power as each other. The Church-Turing Thesis asserts that this formal model, i.e., "Turing computability", exactly captures the intuitive notion of effective computablity. Here we are ignoring running-time and memory requirements, as long as they are finite. This is a well-believed thesis. Note that it is not provable because it involves an undefined, intuitive notion.

It is possible to encode all of mathematics in the first-order logic of set-theory. Thus we may define the problem of mathematics as follows,

$$\text{MATH} \quad = \quad \left\{ \varphi \in \mathcal{L}(\Sigma_{\text{set}}) \;\middle|\; \varphi \text{ is true} \right\} .$$

Hilbert believed that MATH is decidable and he wanted logicians to prove this.

Definition 12.5 [recursive enumerability] $S \subseteq \{0,1\}^*$ is *recursively enumerable* (r.e.) if and only if its *partial* or *polite characteristic function* is computable:

$$P_S(w) = \begin{cases} 1 & \text{if } w \in S \\ \nearrow & \text{if } w \notin S \end{cases}$$

The symbol, " $\nearrow$ ", means undefined. Thus, if a program, M, computes P_S and $w \in S$, then on input w, M will eventually halt and output 1. However, if $w \notin S$, then M will never answer. □

Note that recursive and recursively enumerable are properties of problems. We can also define the sets of all recursive and recursively enumerable problems. These are the complexity classes at the top of our diagram of computability and complexity:

$$\begin{aligned} \text{Recursive} &\stackrel{\text{def}}{=} \{S \subseteq \{0,1\}^* \mid S \text{ is recursive}\} \\ \text{r.e.} &\stackrel{\text{def}}{=} \{S \subseteq \{0,1\}^* \mid S \text{ is recursively enumerable}\} \end{aligned}$$

Lemma 12.6 Recursive $\subseteq$ r.e.

Proof: We need to show that for all S, $S \in$ Recursive $\Rightarrow S \in$ r.e.. Suppose that $S \in$ Recursive. This means that χ_S is computed by some Turing Machine M_S. We shall use M_S to construct a new Turing Machine M'_S which computes P_S.

$$M'_S(x) \quad := \quad \textbf{if } (M_S(x) == 1) \quad \textbf{then return}(1) \quad \textbf{else run forever}$$

□

Definition 12.7 [co-r.e.] The complementary class of r.e. is co-r.e. $\stackrel{\text{def}}{=} \{S \mid \overline{S} \in \text{r.e.}\}$. □

Theorem 12.8 Recursive $=$ r.e. $\cap$ *co*-r.e..

Proof: We have already seen that Recursive $\subseteq$ r.e.. We can similarly show that Recursive $\subseteq$ co-r.e.. Therefore, Recursive $\subseteq$ r.e. $\cap$ co-r.e.. We now show the other direction: r.e. $\cap$ co-RE $\subseteq$ Recursive.

Suppose that $S \in$ r.e. $\cap$ co-r.e.. Let TM's M_1, M_0 compute $P_S, P_{\overline{S}}$, respectively. Define TM M_S which does the following: On input, w, run $M_1(w)$ and $M_0(w)$ in parallel. If $M_1(w)$ ever halts and returns 1, then return(1). If $M_0(w)$ ever halts and returns 1, then return(0). Note that for every w, exactly one of these events will occur. Thus M_S computes χ_S. Thus $S \in$ Recursive. □

12.1 Constructing a set K which is r.e. but not Recursive.

Note that every Java program, or equivalently every TM, can be encoded by a binary string. We can thus list them out in lexicographic order: all valid TM's encoded in 1 bit, all valid TM's encoded in 2 bits, all valid TM's encoded in 3 bits, ... We write this listing as $M_0, M_1, M_2, \cdots$.

Remark 12.9 It is a rather interesting and powerful idea that all the possible algorithmic ideas of all time are on this list: $M_0, M_1, M_2, \cdots$. □

Even though it is now considered obvious, on of the most interesting and important theorems in computer science was proved by Turing in his original 1936 paper:

Theorem 12.10 (Universal Turing Machine [Turing, 1936]) *There exists a universal TM, U, such that for all programs $n \in \mathbf{N}$ and inputs $w \in \mathbf{N}$,* $\quad U(n,w) = M_n(w), \quad$ *i.e. on input (n,w), U computes exactly what the nth TM, M_n, does on input w.*

Definition 12.11 [W_i, the ith r.e. set] For each TM, M_i, define W_i, the set of strings accepted by TM M_i,

$$W_i \stackrel{\text{def}}{=} \mathcal{L}(M_i) \quad = \quad \{w \mid M_i(w) = 1\} \, .$$

□

Proposition 12.12 *A set, S, is* r.e. *iff $S = W_i$ for some $i \in$* **N**.

To construct a set that is r.e. but not recursive, Turing used diaglonalization (the method that Cantor used to prove that **R** is uncountable).

Let us list out all the r.e. sets graphically. Instead of thinking of the sets as containing binary strings, think of them as containing natural numbers. Thus we list a set, W_i, by an infinite sequence of 0's and 1's indicating whether $0 \in W_i, 1 \in W_i, \ldots$.

n	0	1	2	3	4	5	6	7	8	$\cdots$	W_n
0	$\boxed{0}$	0	0	0	0	0	0	0	0	$\cdots$	W_0
1	1	$\boxed{1}$	1	1	1	1	1	1	1	$\cdots$	W_1
2	1	0	$\boxed{1}$	0	1	0	1	0	1	$\cdots$	W_2
3	0	1	0	$\boxed{1}$	0	1	0	1	0	$\cdots$	W_3
4	1	0	0	0	$\boxed{0}$	0	0	0	0	$\cdots$	W_4
5	0	0	1	1	0	$\boxed{1}$	0	1	0	$\cdots$	W_5
6	1	0	0	1	0	0	$\boxed{1}$	0	0	$\cdots$	W_6
7	1	1	0	0	0	0	0	$\boxed{0}$	0	$\cdots$	W_7
8	0	1	0	0	0	0	0	0	$\boxed{0}$	$\cdots$	W_8
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
	0	1	1	1	0	1	1	0	0	$\cdots$	K
	1	0	0	0	1	0	0	1	1	$\cdots$	$\overline{K}$

Define the diagonal set and its complement:

$$\begin{array}{rcccl} K & = & \{n \mid M_n(n) = 1\} & = & \{n \mid n \in W_n\} \\ \overline{K} & = & \{n \mid M_n(n) \neq 1\} & = & \{n \mid n \notin W_n\} \end{array}$$

Proposition 12.13 $\quad \overline{K} \in$ *co*-r.e. $-$ r.e.

Proof: K is r.e. since P_K is computed by the TM which on input n simulates $M_n(n)$ and returns 1 iff $M_n(n)$ returns 1. Thus $\overline{K}$ is co-r.e..

Suppose for the sake of a contradiction that $\overline{K} \in$ r.e.. Then by Prop. 12.12, $\overline{K} = W_c$ for some $c \in \mathbf{N}$. However, by the definition of $\overline{K}$,

$$c \in \overline{K} \;\Leftrightarrow\; c \notin W_c \;\Leftrightarrow\; c \notin \overline{K} \, .$$

This is a contradiction. Thus, $\overline{K}$ is not r.e. □

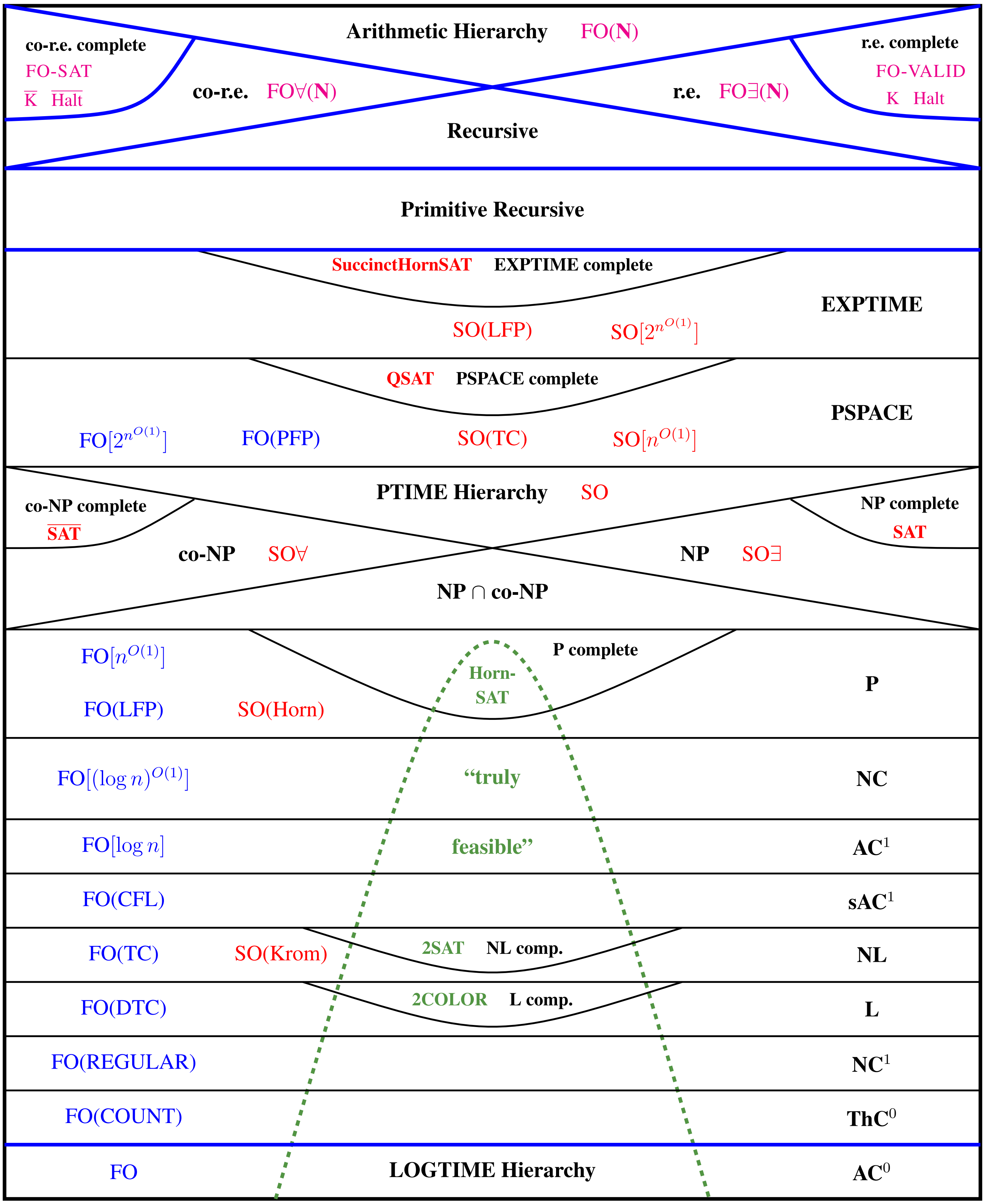
Define the halting problem HALT $\stackrel{\text{def}}{=} \{(n,w) \mid M_n(w)\downarrow\}$ where $M_n(w)\downarrow$ means that TM M_n on input w eventually converges, i.e., halts and thus provides an output. Thus,

$$M_n(x)\downarrow \quad \Leftrightarrow \quad M_n(x) \in \mathbf{N} \quad \Leftrightarrow \quad M_n(x) \neq \nearrow$$

Theorem 12.14 (Unsolvability of the halting problem) HALT *is not recursive.*

Proof: Suppose for the sake of a contradiction that HALT were recursive and the TM M_H computes χ_{HALT}. Then, using M_H we can build a TM to compute K as follows: on input n run $M_H(n,n)$. If it says 1, then we know that $M_n(n)$ eventually halts, so run it. If it returns 1 then return 1, otherwise return 0. If $M_H(n,n) = 0$ then $n \notin K$, so return 0. Since K is not recursive, neither is HALT. □

Remark 12.15 To Hilbert's great disappointment, Church, Gödel and Turing proved that MATH is not decidable and in fact, MATH $\notin$ r.e.. Why does this follow from Thm. 12.14? □



Arithmetic Hierarchy
$FO(\mathbf{N})$
co-r.e. complete
FO-SAT
$\overline{\mathrm{K}}$ $\overline{\mathrm{Halt}}$
co-r.e.
$FO\forall(\mathbf{N})$
r.e. complete
FO-VALID
K Halt
r.e.
$FO\exists(\mathbf{N})$
Recursive
Primitive Recursive
SuccinctHornSAT
EXPTIME complete
EXPTIME
SO(LFP)
$SO[2^{n^{O(1)}}]$
QSAT
PSPACE complete
PSPACE
$FO[2^{n^{O(1)}}]$
FO(PFP)
SO(TC)
$SO[n^{O(1)}]$
PTIME Hierarchy
SO
co-NP complete
$\overline{\mathrm{SAT}}$
NP complete
SAT
co-NP
$SO\forall$
NP
$SO\exists$
NP ∩ co-NP
$FO[n^{O(1)}]$
P complete
Horn-SAT
P
FO(LFP)
SO(Horn)
$FO[(\log n)^{O(1)}]$
“truly
NC
$FO[\log n]$
feasible”
AC^1
FO(CFL)
sAC^1
FO(TC)
SO(Krom)
2SAT
NL comp.
NL
FO(DTC)
2COLOR
L comp.
L
FO(REGULAR)
NC^1
FO(COUNT)
ThC^0
FO
LOGTIME Hierarchy
AC^0