計算的極限

不可判定性與 Entscheidungsproblem 學習教材

5CCS2FC2 Foundations of Computing II — Week 10 白話講義

根據 Dr Christopher Hampson（King's College London）之投影片整理與擴充

2026 年 7 月

本講義在講什麼？

這份教材對應 Week 10 的兩份投影片：

1. **week10_undecidable.pdf**：什麼是「可判定的語言」？是否所有問題都能用電腦解決？如何把圖靈機本身編碼成字串，並造出一台能模擬所有機器的「萬能圖靈機」？
2. **week10_entscheidungsproblem.pdf**：希爾伯特提出的「判定問題」（Entscheidungsproblem）——是否存在一個演算法，能判斷任何邏輯公式是否恆真？答案是「不存在」，而本講義將白話解釋為什麼。

一句話總結本週主題：電腦不是萬能的——有些問題在數學上被證明「永遠寫不出解決它的程式」。

Contents

1 背景故事：希爾伯特的夢想與它的破滅

要理解本週的內容，最好先從一段歷史說起。

二十世紀初，德國大數學家希爾伯特（**David Hilbert**）有一個宏偉的夢想：把整個數學「機械化」。他相信數學裡的每一個問題都有答案，而且我們應該能找到一套固定的程序（演算法），只要照著步驟做，就能判斷任何數學敘述是真是假。1928 年，他和學生阿克曼（Wilhelm Ackermann）正式提出了著名的判定問題（**Entscheidungsproblem**，德文「決定問題」的意思），並稱之為「數理邏輯的核心問題」。

白話比喻：數學的「自動販賣機」

希爾伯特想要的東西，就像一台數學自動販賣機：你把任何一條數學敘述（例如「哥德巴赫猜想」）投進去，機器咔啦咔啦運轉一陣子，然後保證吐出「真」或「假」的答案。如果這台機器存在，數學家就可以退休了——所有未解難題都只是「還沒排隊進機器」而已。

然而這個夢想接連遭受兩記重擊：

1. **1931** 年，哥德爾（**Kurt Gödel**）的不完備定理：任何足夠強大且一致的數學公理系統中，必然存在「為真卻無法被證明」的敘述。這說明「真」和「可證明」不是同一回事。
2. **1936** 年，邱奇（**Alonzo Church**）與圖靈（**Alan Turing**）分別獨立證明：判定問題無解——根本不存在希爾伯特想要的那種通用判定程序。邱奇用的是 λ-演算；圖靈則發明了「圖靈機」這個計算模型，並透過「停機問題」證明了同樣的結論。

圖靈 1936 年的論文《On Computable Numbers, with an Application to the Entscheidungsproblem》（論可計算數及其在判定問題上的應用）被譽為計算機科學史上最重要的論文之一——它一口氣引入了圖靈機、萬能圖靈機，以及不可判定問題的存在。第二份投影片的最後一頁，放的正是這篇論文的首頁。

為什麼要先講歷史？

因為本週兩份投影片其實是同一個故事的兩半：第一份投影片建立工具（圖靈機的編碼、萬能圖靈機、不可判定語言的存在性），第二份投影片用這些工具完成最後一擊——證明希爾伯特的判定問題無解。

2 Church–Turing 論題：什麼叫「可以被計算」？

第一份投影片開頭就給出這個命題：

Church–Turing 論題（Church–Turing Thesis）

任何能夠透過某個有限程序（**finite process**）有效計算的語言，都可以被某台圖靈機辨識。

2.1 白話解釋

「有限程序」指的就是我們直覺上的「演算法」：一份步驟明確、不需要靈感或運氣、任何人（或機器）照做都會得到相同結果的操作說明書。Church–Turing 論題主張：

> 凡是人類直覺上「可以一步一步算出來」的東西，圖靈機都算得出來。

這意味著圖靈機不只是眾多計算模型之一，而是「計算」這個概念本身的數學定義。你的手機、超級電腦、Python 程式，計算能力都不超過一台圖靈機（頂多比較快而已）。

注意：這是「論題」不是「定理」

Church–Turing 論題無法被數學證明，因為「直覺上可計算」不是一個精確的數學概念——它連接的是「非正式的直覺」與「正式的數學模型」。但它獲得了壓倒性的證據支持：邱奇的 λ-演算、圖靈機、遞迴函數、以及後來所有的程式語言，被證明計算能力全部相等。近百年來沒有人找到反例。

2.2 它的反面意義才是本週重點

Church–Turing 論題有一個殺傷力極強的推論：如果我們證明「沒有任何圖靈機能解決問題 **X**」，那就等於證明「沒有任何演算法、任何程式語言、任何未來的電腦能解決問題 **X**」。這不是工程限制，而是數學上的不可能，就像「畫一個四邊形的三角形」一樣不可能。本週接下來的所有內容，都是在為這種「不可能性證明」鋪路。

3 可判定語言：健全、完備、會停機

3.1 記號約定

投影片先定義了一個方便的記號。對於圖靈機 M 與輸入字串 w：

$$M(w) := \begin{cases} 1 & \text{若} M \text{ 接受（accept）} w \\ 0 & \text{若} M \text{ 拒絕（reject）} w \\ \uparrow & \text{若} M \text{ 在輸入} w \text{ 上永不停機} \end{cases}$$

第三種情況 $\uparrow$（讀作「發散」或「卡住」）非常關鍵：圖靈機跟真實程式一樣，可能陷入無窮迴圈，永遠跑不完。任何寫過 `while True:` 的人都懂這種感覺。

3.2 可判定（decidable）的三個條件

定義：可判定語言

語言 L 是可判定的（**decidable**），如果存在某台圖靈機 M 同時滿足：

- 健全（**Sound**）：若 $M(w) = 1$，則 $w \in L$。——「M 說是，就真的是」，不會誤報。
- 完備（**Complete**）：若 $w \in L$，則 $M(w) = 1$。——「真的是，M 就會說是」，不會漏報。
- 會停機（**Terminating**）：對所有輸入 $w \in \Sigma^*$，M 的執行時間 $T_M(w) < \infty$。——不管餵什麼，M 都會在有限時間內給出答案。

範例：三個條件各自失守會怎樣

想像一個「檢查數字是否為質數」的程式：

- 不健全：它說 9 是質數（誤報）。這種程式即使會停也不可信。
- 不完備：它對 7 說「不是質數」（漏報）。同樣不可信。
- 不停機：它對 7 正確回答、對 9 正確回答，但你輸入 11 的時候它永遠轉圈圈。你等了一小時沒答案——問題是你永遠不知道該再等五分鐘，還是它根本不會回來。

三者同時成立，才能叫「判定」了這個語言。前兩者合起來的意思是「答案永遠正確」，第三者的意思是「答案永遠會出現」。

可判定 vs. 可辨識：差一個「會停機」

如果 M 只滿足健全與完備，但對於不在 L 裡的字串可能永不停機，我們稱 L 是可辨識的（**recognisable**，又稱 recursively enumerable）。直覺：可辨識 =「答案是 Yes 的時候你終究會知道；答案是 No 的時候你可能等到天荒地老」。可判定 =「不管 Yes 或 No 你都會在有限時間內知道」。本週的重頭戲——停機問題——正是「可辨識但不可判定」的經典例子。

3.3 所有語言都可判定嗎？

投影片在這裡拋出關鍵問題：**Is Every Language Decidable?** 答案是否定的，而且有一個優雅的「數數」論證：

計數論證：不可判定語言必然存在

- 每台圖靈機都可以用一個有限長度的字串描述（下一節會示範怎麼編碼）。有限字串的集合是可數無窮的（可以跟 $1, 2, 3, \ldots$ 一一對應），所以「圖靈機的總數」是可數無窮。
- 但「語言」是字串的任意子集合，即 Σ^* 的冪集。康托（Cantor）的對角線論證告訴我們，冪集的基數嚴格更大——語言的總數是不可數無窮。
- 機器（可數多台）不夠分給語言（不可數多個），所以絕大多數的語言根本沒有對應的圖靈機，遑論判定。

白話比喻：旅館房間不夠

把每台圖靈機想成一間旅館房間（房號 1, 2, 3, …可以編號下去），把每個語言想成一位想住房的客人。康托證明了：客人的數量是「更高一級的無窮多」，無論怎麼安排，都必然有（而且是絕大多數）客人分不到房間。這些「沒房間的客人」就是無法被任何程式解決的問題。

不過，計數論證只告訴我們不可判定的語言存在，卻沒有指出哪一個語言不可判定。要找出一個具體的例子（例如停機問題），我們需要先學會「把圖靈機餵給圖靈機」——這就是接下來編碼與萬能圖靈機的主題。

4 把一切編碼成字串：程式也是資料

4.1 編碼字串組（tuples）

投影片示範：多個（非空）字串組成的序列 $(w_0, w_1, \ldots, w_n)$，可以用一個擴充字母表 $\Sigma \cup \{\#\}$ 上的單一字串來表示：

$$\langle w_0, w_1, \ldots, w_n \rangle := \# \, w_0 \, \# \, w_1 \, \# \cdots \# \, w_n \, \#$$

也就是用特殊符號 # 當「分隔欄」。如果不想擴充字母表，也可以把每個字元轉寫成二進位編碼，例如 $0 \mapsto 00$、$1 \mapsto 01$、$\# \mapsto 11$，這樣整個 tuple 就變成一條純 0/1 字串。

範例

$\langle 01, 10 \rangle = \#01\#10\#$。若再轉成二進位編碼：$\#01\#10\# \mapsto 11\,00\,01\,11\,01\,00\,11$。這就像 CSV 檔用逗號分隔欄位，或網路封包用固定格式打包多筆資料——概念完全相同。

4.2 編碼有限函數

有限函數 $f : A \to B$（A, B 為有限集合）可以編碼成「tuple 的 tuple」：

$$\langle f \rangle := \big\langle \, \langle a_1, f(a_1) \rangle, \, \langle a_2, f(a_2) \rangle, \, \ldots, \, \langle a_n, f(a_n) \rangle \, \big\rangle$$

效果上等於把 f 存成一張 $2 \times n$ 的查詢表。這跟程式裡把函數存成字典（`{a1: b1, a2: b2, ...}`）是一樣的想法。

4.3 編碼整台圖靈機

關鍵一步：圖靈機本身就是「幾個狀態加上一個有限的轉移函數 δ」，而這些全都是有限物件，所以整台圖靈機也能編碼成一條字串：

$$\text{code}(M) := \big\langle \, q_{\text{init}}, \, q_{\text{accept}}, \, q_{\text{reject}}, \, \langle \delta \rangle \, \big\rangle$$

這一步的哲學意義：程式 = 資料

「機器可以被寫成字串」聽起來平凡，卻是整個現代計算的基石：

- 你的 Python 程式就是一個 `.py` 文字檔——程式是資料。
- 編譯器是「讀入程式、輸出程式」的程式——程式可以操作程式。
- 也因此，一台機器可以把另一台機器的描述當作輸入來分析或模擬——這正是萬能圖靈機與所有不可判定性證明的前提。

5 萬能圖靈機：一台機器模擬所有機器

定義：萬能圖靈機（Universal Turing Machine）

萬能圖靈機 M_u 是一台接受配對輸入 $\langle \text{code}(M), w\rangle$ 的圖靈機，滿足：

$$\begin{aligned} M_u \text{ 接受} \langle \text{code}(M), w\rangle &\iff M \text{ 接受} w \\ M_u \text{ 拒絕} \langle \text{code}(M), w\rangle &\iff M \text{ 拒絕} w \\ M_u \text{ 停機於} \langle \text{code}(M), w\rangle &\iff M \text{ 停機於} w \end{aligned}$$

簡寫成一行：$M_u\big(\langle \text{code}(M), w\rangle\big) = M(w)$。

5.1 白話解釋：它就是直譯器

投影片說得很直接：萬能圖靈機的行為就像編譯器或直譯器——它讀入「軟體」$\text{code}(M)$，然後在輸入 w 上模擬 M 的一舉一動。M 接受它就接受、M 拒絕它就拒絕、M 卡住它也跟著卡住（注意最後一點：M_u 並不能「預知」M 會卡住而提早喊停——它只能傻傻跟著跑）。

現代對照：你每天都在用萬能圖靈機

- **Python 直譯器**：讀入 `program.py`（$= \text{code}(M)$）和使用者輸入（$= w$），模擬程式執行。
- 遊戲模擬器：一台 PC 模擬整台紅白機的硬體行為。
- 虛擬機/雲端：同一台實體伺服器輪流「扮演」上千台不同的虛擬電腦。

1936 年圖靈在紙上證明萬能機器存在，等於預言了「通用電腦」的可行性：我們不需要為每個任務造一台專屬機器，只要造一台通用機器，再把不同的「軟體」餵給它。這就是「儲存程式型電腦」（stored-program computer）概念的理論起源。

5.2 M_u 大致如何運作？

概念上，M_u 的磁帶劃分成三個工作區：(1) 存放 $\text{code}(M)$，即 M 的轉移表；(2) 記錄 M 目前的狀態；(3) 維護 M 的磁帶內容與讀寫頭位置。每模擬一步，M_u 就去轉移表裡「查表」：目前

狀態、目前符號 → 該寫什麼、往哪移、換什麼狀態，然後更新工作區。就像人肉執行程式碼時，用一張草稿紙追蹤變數值一樣。

6 停機問題：第一個具體的不可判定問題

有了「機器可編碼」與「機器可模擬機器」這兩項武器，我們就能建構出一個具體的不可判定問題。這是連接兩份投影片的橋樑（第二份投影片的證明會直接引用它）。

接受問題與停機問題

- 接受問題（**Accepting Problem**） $A_{TM} = \{\langle \text{code}(M), w\rangle : M \text{ 接受} w\}$
- 停機問題（**Halting Problem**） $H_{TM} = \{\langle \text{code}(M), w\rangle : M \text{ 在} w \text{ 上會停機}\}$

定理（**Turing, 1936**）：兩者皆不可判定。

6.1 為什麼不能「直接跑跑看」?

一個自然的想法：想知道 M 會不會停，就用萬能圖靈機模擬它啊！問題在於——如果 M 停了，你確實得到答案；但如果 M 不會停，你會陪它一起跑到永遠，永遠等不到「它不會停」這個答案。模擬只能給你「可辨識」，給不了「可判定」。判定需要的是對兩種答案都在有限時間內回覆。

6.2 對角線證明（白話版）

證明梗概：自我指涉製造矛盾

用反證法。假設存在一台會停機的判定器 H：

$$H(\langle \text{code}(M), w\rangle) = \begin{cases} 1 & \text{若} M \text{ 在} w \text{ 上停機} \\ 0 & \text{若} M \text{ 在} w \text{ 上不停機} \end{cases}$$

利用 H 造一台「唱反調機」D，它吃一台機器的編碼 $\text{code}(M)$，然後：

1. 呼叫 H 詢問：「M 吃自己的編碼當輸入時，會停嗎 ?」
2. 若 H 回答「會停」$\Rightarrow$ D 故意進入無窮迴圈；
3. 若 H 回答「不停」$\Rightarrow$ D 立刻停機。

最後的殺招：把 D 餵給它自己，考慮 $D(\text{code}(D))$：

- 若 D 吃自己會停 $\Rightarrow$ 依定義第 2 步，D 進入無窮迴圈 $\Rightarrow$ 不停。矛盾！
- 若 D 吃自己不停 $\Rightarrow$ 依定義第 3 步，D 立刻停機 $\Rightarrow$ 停。矛盾！

兩種情況都矛盾，所以假設錯誤：判定器 H 不存在。 ■

白話比喻：理髮師悖論

這個證明的邏輯結構跟著名的「理髮師悖論」一模一樣：小鎮理髮師宣稱「我只幫不自己刮鬍子的人刮鬍子」。那理髮師自己的鬍子誰刮？他若自己刮，就違反了「只幫不自己刮的人刮」；他若不自己刮，那依規則他就該幫自己刮。規則在「套用到自己身上」時自爆——所以這樣的理髮師根本不可能存在。同理，H 在被 D「套用到自己身上」時自爆，所以 H 不可能存在。

注意這裡「程式吃自己的原始碼」一點都不奇幻：編譯器編譯自己的原始碼（bootstrapping）、語法檢查器檢查自己的程式碼，都是業界日常。

常見誤解澄清

- 不可判定 $\neq$「所有情況都答不出來」。對很多特定程式，我們當然看得出會不會停（例如空迴圈）。不可判定的意思是：不存在一個對所有程式–輸入組合都正確且會停機的通用判定演算法。
- 不可判定 $\neq$「目前技術不夠」。這是數學證明的不可能，再快的電腦、再聰明的 AI 都無法突破——除非它用的計算模型超越圖靈機（依 Church–Turing 論題，目前沒有已知的物理裝置做得到）。

7 Entscheidungsproblem：判定問題的殞落

現在進入第二份投影片的主題。先把問題本身說清楚：

Entscheidungsproblem（判定問題）

輸入：一條謂詞邏輯（predicate logic，即一階邏輯）公式 F。
輸出：若且唯若 F 是恆真式（**tautology**）時輸出 True。

白話說：希爾伯特想要一個演算法，餵它任何一條一階邏輯公式，它都能正確判斷「這條公式是否在所有情況下皆為真」。

為什麼命題邏輯可以，謂詞邏輯卻不行？

注意對比：命題邏輯的恆真式判定是可判定的——公式只有有限多個變數，把真值表全列出來檢查即可（頂多指數時間，但保證會停）。謂詞邏輯多了 $\forall$、$\exists$ 量詞，變數的取值範圍可以是無窮的論域，真值表法失效。正是這個「無窮」讓判定問題落入不可判定的深淵——下面的證明會準確揭示無窮是怎麼混進來的。

定理（Church 1936；Turing 1936）

Entscheidungsproblem 是不可判定的。

7.1 證明策略：歸約（reduction）

投影片的證明採用「歸約」策略，這是證明不可判定性的標準武器：

> 如果「已知無解的問題 A」可以改寫成「問題 B」，那 B 也必然無解。
> （否則解 B 的演算法就能拿來解 A，矛盾。）

這裡的 A 是上一節的接受問題（M 是否終將接受 w？——與停機問題同樣不可判定），B 是 Entscheidungsproblem。我們要示範：任何「機器 M 是否接受 w」的問題，都能機械化地改寫成一條邏輯公式的判定問題。投影片把證明分成四步。

7.2 Step 1：回顧 Cook–Levin 的公式建構

在 Cook–Levin 定理（證明 SAT 是 NP-complete 的那個定理）的證明中，我們已經學過一項技術：給定機器 M 與輸入 w，可以建構一條命題邏輯公式 $F_{M,w}$，使得

$$F_{M,w} \text{ 可滿足} \iff M \text{ 在多項式步數內接受} w.$$

做法是用一堆布林變數把「計算過程的快照」寫下來：哪個時間點、磁帶哪一格是什麼符號、讀寫頭在哪、機器處於哪個狀態，再用邏輯條件強制「每一步都遵守 M 的轉移規則」。整個計算歷史被壓縮成一張巨大的表格，公式 $F_{M,w}$ 就是「這張表格是一次合法且成功的計算」的邏輯描述。

7.3 Step 2：命題邏輯的天花板——寫不出「終將」

問題來了。Cook–Levin 的建構之所以可行，是因為我們事先知道計算最多跑多項式步，所以變數的數量有限。但接受問題沒有步數上限——我們想表達的是：

> 「機器 M 終將（eventually）接受」——不管要跑多少步。

用命題邏輯寫出來會變成一條無限長的「或」：

$$S_{q,0} \vee S_{q,1} \vee S_{q,2} \vee S_{q,3} \vee S_{q,4} \vee S_{q,5} \vee S_{q,6} \vee \cdots$$

（其中 $q = q_{\text{accept}}$，$S_{q,t}$ 表示「時刻 t 機器處於接受狀態」。）命題邏輯的公式必須是有限長的，所以命題邏輯表達不了「終將接受」這種全域性質。這就是為什麼 SAT 可判定（甚至只是 NP-complete），接受問題卻不可判定——差別就在這個「無窮」。

7.4 Step 3：謂詞邏輯的量詞恰好補上這個洞

謂詞邏輯的 $\exists$ 量詞正是為「無限選擇」而生的。選取適當的謂詞符號，把「時間」從變數的下標升級成謂詞的參數：

$$\begin{aligned} C_a(i,t) &= \text{「時刻}t\text{ 時，磁帶第}i\text{ 格的內容是符號}a\text{」} \\ H(i,t) &= \text{「時刻}t\text{ 時，讀寫頭位於第}i\text{ 格」} \\ S_q(t) &= \text{「時刻}t\text{ 時，機器處於狀態}q\text{」} \end{aligned}$$

那條寫不完的無限「或」就能收斂成短短一句：

$$S_q(0) \vee S_q(1) \vee S_q(2) \vee S_q(3) \vee \cdots \quad \rightsquigarrow \quad \exists t\, S_q(t)$$

——「存在某個時刻 t，機器處於接受狀態」。一個量詞就馴服了無窮。

白話比喻：從「點名」到「一句話」

命題邏輯像是只能逐一點名：「第 0 秒接受了嗎？第 1 秒接受了嗎？第 2 秒…」——永遠點不完。謂詞邏輯則可以直接說一句「總有一天它會接受」。表達力的提升讓公式變短了，但代價是：這麼強的語言，再也沒有演算法能全面判定它的真假。表達力與可判定性之間存在根本的取捨（trade-off），這是邏輯學反覆出現的主旋律。

7.5 Step 4：完成歸約

於是，仿照 Cook–Levin 的建構、但改用上述謂詞符號（並把「遵守轉移規則」等條件也用 $\forall i\,\forall t$ 寫成有限長的公式），我們可以對任意 M 與 w 機械化地寫出一條謂詞邏輯公式 $F_{M,w}$，使得：

$$F_{M,w} \text{ 可滿足} \iff M \text{ 終將接受輸入} w. \qquad \text{Q.E.D.}$$

投影片的收尾註解點出了整個論證的骨架：這實際上是一個從接受問題到 **Entscheidungsproblem** 之補問題的多項式歸約。梳理一下這句話：

- 「F 可滿足」等價於「$\neg F$ 不是恆真式」。所以「可滿足性判定」正是「恆真性判定」的補問題（答案互為顛倒）。
- 若 Entscheidungsproblem 可判定，那可滿足性也可判定（問 $\neg F$ 是否恆真、再把答案反過來即可）。
- 那麼接受問題也可判定了：給定 $\langle M, w\rangle$，造出 $F_{M,w}$，判定其可滿足性，就知道 M 是否終將接受 w。
- 但接受問題已被證明不可判定（上一節的對角線論證）。矛盾！

所以 Entscheidungsproblem 不可判定。希爾伯特的自動販賣機，永遠造不出來。

整條證明鏈一圖看懂

對角線論證 $\Longrightarrow$ 接受 / 停機問題不可判定 $\Longrightarrow$
（把計算歷史編碼成謂詞邏輯公式）$\Longrightarrow$ Entscheidungsproblem 不可判定

第一份投影片提供左半邊的工具（編碼、萬能機、不可判定語言的存在），第二份投影片完成右半邊的歸約。

7.6 圖靈的原始論文

第二份投影片最後附上了圖靈 1936 年論文的首頁。這篇論文於 1936 年 5 月 28 日收稿、11 月 12 日宣讀，當年圖靈只有 24 歲。論文開頭定義了「可計算數」——小數展開能以有限手段算出的實數——並在文末證明判定問題無解。有趣的是，邱奇早幾個月用 λ-演算發表了相同結論，但圖靈的證明因為建立在「機器」這個直觀模型上，被普遍認為更有洞察力，也直接催生了現代電腦科學。

8 那又如何？不可判定性的現實影響

這些 1936 年的定理，至今每天都在影響軟體工程：

- 沒有完美的無窮迴圈偵測器：IDE 或編譯器永遠不可能對所有程式正確警告「這段程式碼會卡死」。（停機問題）
- 沒有完美的防毒軟體：「判斷任意程式是否具有惡意行為」可歸約到停機問題，因此任何防毒引擎必然有誤報或漏報。
- 程式驗證的極限：萊斯定理（**Rice's Theorem**，本課程的自然延伸）說：關於程式行為的任何非平凡性質（會不會輸出 0？功能是否等同另一支程式？）通通不可判定。這就是為什麼自動化測試不能證明程式無誤，只能找到部分錯誤。
- 編譯器最佳化的極限：「這兩段程式碼是否等價」不可判定，所以最佳化只能靠保守的、可判定的近似規則。

工程上的正確心態

不可判定不代表躺平。實務的出路是：（1）解受限版本的問題（如只分析不含遞迴的程式）；（2）接受近似答案（允許回答「不知道」，如靜態分析工具）；（3）設超時上限（跑超過 10 秒就當作可能不會停）。理解極限，才能聰明地繞過極限。

9 重點整理與自我測驗

9.1 一頁複習

概念	一句話白話文
Church–Turing 論題	直覺上「算得出來」的東西，圖靈機都算得出來；圖靈機是「計算」的官方定義。
可判定語言	存在健全（不誤報）、完備（不漏報）且對所有輸入都會停的機器。
可辨識語言	Yes 的時候終究會告訴你；No 的時候可能永遠沒有下文。
編碼 code(M)	機器本身可以寫成字串——程式就是資料。
萬能圖靈機 M_u	一台「直譯器」機器，讀入 $\langle \text{code}(M), w \rangle$ 便模擬 $M(w)$。
計數論證	機器可數無窮、語言不可數無窮，所以大多數語言無機器可解。
停機 / 接受問題	「任意程式在任意輸入上會不會停 / 接受」——用對角線自我指涉證明不可判定。
Entscheidungsproblem	「任意謂詞邏輯公式是否恆真」——把接受問題的計算歷史編碼成公式而歸約，故不可判定。
$\exists t\, S_q(t)$	量詞把命題邏輯寫不出的無限「或」壓縮成一句「總有一天」。

9.2 自我測驗

1. 請分別說明「健全」「完備」「會停機」三個條件，並各舉一個「只違反其中一個條件」的程式例子。
2. 可判定與可辨識差在哪裡？停機問題屬於哪一類？為什麼「用萬能圖靈機直接模擬」只能證明可辨識？
3. 為什麼「圖靈機的數量」是可數的，而「語言的數量」是不可數的？這個對比推出什麼結論？
4. 用理髮師悖論的語言，重述停機問題不可判定的證明。「唱反調機」D 對應悖論中的哪個角色？
5. 命題邏輯版的 $F_{M,w}$（Cook–Levin）與謂詞邏輯版的 $F_{M,w}$ 差在哪裡？為什麼前者只能表達「多項式步數內接受」而後者能表達「終將接受」？
6. 「F 可滿足 $\iff$ $\neg F$ 不是恆真式」在歸約中扮演什麼角色？為什麼投影片說這是歸約到

Entscheidungsproblem 的補問題？

7. （思考題）SAT 是 NP-complete（很難但可判定），Entscheidungsproblem 卻不可判定。兩者的建構幾乎相同，差別到底出在哪一個環節？

9.3 參考解答提示

1. 健全 = 說 Yes 就真的是；完備 = 真的是就會說 Yes；停機 = 任何輸入都在有限時間內回答。例：把所有輸入都接受的機器對「質數語言」而言完備但不健全。
2. 可判定要求兩面都會停；停機問題是可辨識但不可判定；模擬在「不停機」的情況下永遠等不到答案，所以只能覆蓋 Yes 的那一面。
3. 機器可編碼成有限字串（可數）；語言是 Σ^* 的任意子集，由康托對角線法知其不可數；故必有語言無任何機器對應。
4. D 對應「理髮師」：規則套到自己身上時，兩種選擇都違反規則，故規則（判定器 H）不可能存在。
5. 命題邏輯變數只能有限多個，必須事先限定步數；謂詞邏輯用 $\exists t$ 把時間變成無界的量化參數。
6. 恆真性判定器可以透過「問 $\neg F$」變成可滿足性判定器；歸約的目標其實是可滿足性，而它是恆真性（Entscheidungsproblem）的補問題。
7. 差在步數是否有上界：有多項式上界 $\Rightarrow$ 公式有限、可判定（SAT）；無上界 $\Rightarrow$ 需要量詞表達「終將」，落入不可判定。

延伸閱讀

- Turing, A. M. (1936). *On Computable Numbers, with an Application to the Entscheidungsproblem*. Proc. London Math. Soc.——原始論文，前幾節出乎意料地好讀。
- Sipser, M. *Introduction to the Theory of Computation*, Ch. 4–5——可判定性與歸約的標準教科書章節。
- Stanford Encyclopedia of Philosophy: *The Church-Turing Thesis*——論題的歷史與哲學細節。
- Erickson, J. *Algorithms*（線上免費）：*Models of Computation* 章節——包含大量不可判定性的歸約範例。