**The Church-Turing thesis**

**Prove & Disprove**

丘奇-图灵论题（Church-Turing Thesis）是一个关于计算的理论假设，它提出任何有效的计算过程都可以通过图灵机来模拟。

证明丘奇-图灵论题：

丘奇-图灵论题的“证明”实际上是指对这个假设的合理性进行论证和支持。以下是一些支持论题的方法：

1. 历史验证：自图灵和丘奇提出相关概念以来，所有已知的有效计算过程都能被图灵机所模拟。

2. 计算模型等价性：已经证明了许多其他计算模型（如λ演算、递归函数等）与图灵机是等价的，这意味着它们有相同的计算能力。

3. 物理可实现性：目前我们构建的计算机都是基于图灵机的原理，而且能够执行各种各样的计算任务，这也间接支持了丘奇-图灵论题。

然而，这些论证并没有严格地证明丘奇-图灵论题，因为论题本身是关于有效计算的哲学定义，而不是一个数学命题。

推翻丘奇-图灵论题：

要“推翻”丘奇-图灵论题，就需要提出一个有效的计算过程，这个计算过程不能被任何图灵机模拟，或者发现图灵机不能捕捉的某种计算能力。

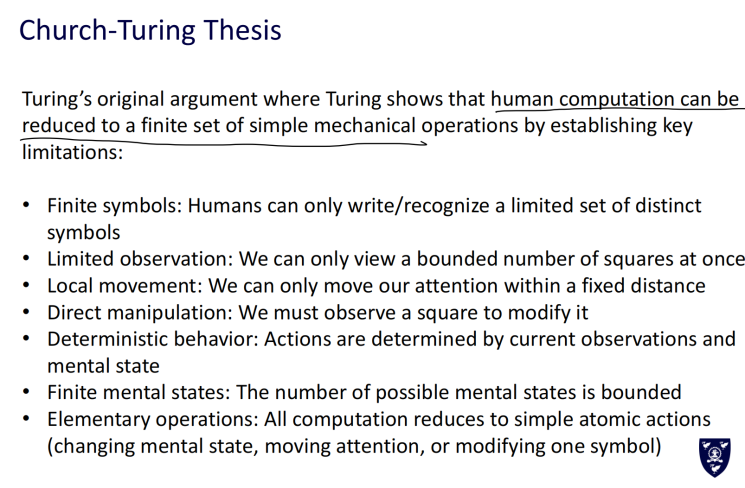
以下是一些理论上可能推翻丘奇-图灵论题的情况：

1. 超计算模型：如果存在一种计算模型能够解决图灵机无法解决的问题，那么这个模型就可以推翻丘奇-图灵论题。例如，如果能够构建出一个可以解决停机问题的机器，那么就推翻了这个论题。

2. 量子计算：虽然量子计算模型（如量子图灵机）并没有直接推翻丘奇-图灵论题，因为它们仍然属于图灵机的扩展，但如果量子计算能够展现出比经典图灵机更强的计算能力（例如，实现量子纠缠和量子叠加带来的速度优势），这可能会对论题的普遍性提出挑战。

3. 物理限制：如果未来的物理学发现某种基本的物理法则允许执行图灵机无法执行的计算，那么这也可能推翻丘奇-图灵论题。

需要注意的是，到目前为止，尽管有许多关于超计算和量子计算的讨论，但丘奇-图灵论题仍然是计算理论中的一个基本假设，并没有被实验或理论所推翻。



## TM Configuration(Lecture 9 p20)

图灵机配置（TM Configuration）

在图灵机的理论中，一个配置（Configuration）是用来描述图灵机在某一时刻的完整状态。

1. 当前状态（State）

用 `q 表示，其中 `q` 是图灵机的状态集合 `Q` 中的一个元素。

2. 磁带内容（Tape Content）：

磁带上的内容可以表示为一个字符串 `uv`，其中 `u` 和 `v` 都是磁带字母表 `Γ` （栈）上的字符串（`Γ\*` 表示 `Γ` 的 Kleene 星，即 `Γ` 的所有可能字符串的集合）。

`u` 表示磁带头左侧的部分，`v` 表示磁带头右侧的部分。

3. 磁带头位置（Tape Head Position）：

磁带头指向的单元格包含字符串 `v` 的第一个符号。

字符串 `uqv`：

- `u`：表示磁带头左侧的所有符号。

- `q`：表示图灵机当前的状态。

- `v`：表示磁带头右侧的所有符号，且磁带头正指向 `v` 的第一个符号。

举例说明

假设有一个图灵机 `M`，其状态集合 `Q` 包含 `{q0, q1, qaccept, qreject}`，磁带字母表 `Γ` 包含 `{0, 1, □}`（其中 `□` 表示空白符号），初始状态为 `q0`，接受状态为 `qaccept`，拒绝状态为 `qreject`。

假设在某一时刻，图灵机的状态为 `q1`，磁带内容为 `01011□□□`，且磁带头指向第一个 `1`。那么这个时刻的图灵机配置可以表示为：

u = 010

q = q1

v = 1□□□

因此，配置字符串为 `010q11□□□`。

配置的作用

- 描述图灵机的状态：通过配置，可以清晰地知道图灵机在某一时刻的完整状态，包括其内部状态和磁带上的内容。

- 模拟图灵机的运行：在图灵机的模拟过程中，通过不断更新配置来表示图灵机的状态变化。

### TM Computation(Lecture 9 p22)

在计算理论中，图灵机的计算是指图灵机在给定输入字符串 `w` 的情况下，从初始配置开始，按照其转换函数 `δ` 的规则，逐步进行状态转换，最终到达一个接受配置的过程。

1. 初始配置（Initial Configuration）：

- 表示为 `C1`，它是图灵机开始计算时的状态。

- `C1` 是由初始状态 `q0` 和输入字符串 `w` 组成的配置，可以写作 `q0w`。

- 这意味着磁带上的内容是输入字符串 `w`，图灵机处于初始状态 `q0`，并且磁带头指向输入字符串的第一个符号。

2. 配置序列（Sequence of Configurations）：

- 从 `C1` 开始，存在一个有限的配置序列 `C1, C2, ..., Ck`，其中 `k` 是一个大于或等于1的整数。

- 每个配置 `Ci` 都按照图灵机的转换函数 `δ` 的规则，转移到下一个配置 `Ci+1`。

3. 转换函数（Transition Function）：

- `δ` 定义了图灵机如何根据当前状态和磁带上的符号进行状态转换。

- 对于每个配置 `Ci`，根据当前状态和磁带头指向的符号，通过 `δ` 函数确定下一个状态、磁带上符号的更改（如果有的话）以及磁头移动的方向。

4. 接受配置（Accepting Configuration）：

- 计算序列的最后一个配置是接受配置，表示为 `Ck`。

- `Ck` 的形式是 `uqacceptv`，其中 `u` 和 `v` 是磁带字母表 `Γ` 上的任意字符串。

- 这意味着图灵机已经转换到接受状态 `qaccept`，表明输入字符串 `w` 被接受。

举例说明

假设我们有一个简单的图灵机 M，其目的是接受所有以1结尾的二进制字符串。

- 状态集合 `Q`：`{q0, q1, qaccept, qreject}`

- 输入字母表 `Σ`：`{0, 1}`

- 磁带字母表 `Γ`：`{0, 1, □}`（`□` 表示空白符号）

- 转换函数 `δ`：

- `δ(q0, 1) = (q1, 1, R)`：在状态 `q0` 遇到 `1`，转移到状态 `q1`，写 `1`，磁头右移。

- `δ(q0, 0) = (q0, 0, R)`：在状态 `q0` 遇到 `0`，保持在状态 `q0`，写 `0`，磁头右移。

- `δ(q1, 1) = (q1, 1, R)`：在状态 `q1` 遇到 `1`，保持在状态 `q1`，写 `1`，磁头右移。

- `δ(q1, 0) = (q1, 0, R)`：在状态 `q1` 遇到 `0`，保持在状态 `q1`，写 `0`，磁头右移。

- `δ(q1, □) = (qaccept, □, R)`：在状态 `q1` 遇到空白符号 `□`，转移到接受状态 `qaccept`，写 `□`，磁头右移。

- 初始状态 `q0`

- 接受状态 `qaccept`

- 拒绝状态 `qreject`

计算过程示例

假设输入字符串 `w = 101`。我们来详细描述图灵机 `M` 的计算过程。

1. 初始配置`C1`：

- 磁带内容：`101□□□...`

- 当前状态：`q0`

- 磁带头位置：指向第一个 `1`

- 配置表示为：`q0101□□□...`

2. 第一步转换 `C1 -> C2`：

- 当前状态：`q0`

- 磁带头指向：`1`

- 根据 `δ(q0, 1) = (q1, 1, R)`：

- 新状态：`q1`

- 写入符号：`1`

- 磁头右移

- 新配置 `C2`：`1q110□□□...`

3. 第二步转换 `C2 -> C3`：

- 当前状态：`q1`

- 磁带头指向：`0`

- 根据 `δ(q1, 0) = (q1, 0, R)`：

- 新状态：`q1`

- 写入符号：`0`

- 磁头右移

- 新配置 `C3`：`10q10□□□...`

4. 第三步转换 `C3 -> C4`：

- 当前状态：`q1`

- 磁带头指向：`1`

- 根据 `δ(q1, 1) = (q1, 1, R)`：

- 新状态：`q1`

- 写入符号：`1`

- 磁头右移

- 新配置 `C4`：`101q1□□□...`

5. 第四步转换 `C4 -> C5`：

- 当前状态：`q1`

- 磁带头指向：`□`

- 根据 `δ(q1, □) = (qaccept, □, R)`：

- 新状态：`qaccept`

- 写入符号：`□`

- 磁头右移

- 新配置 `C5`：`101□qaccept□□□...`

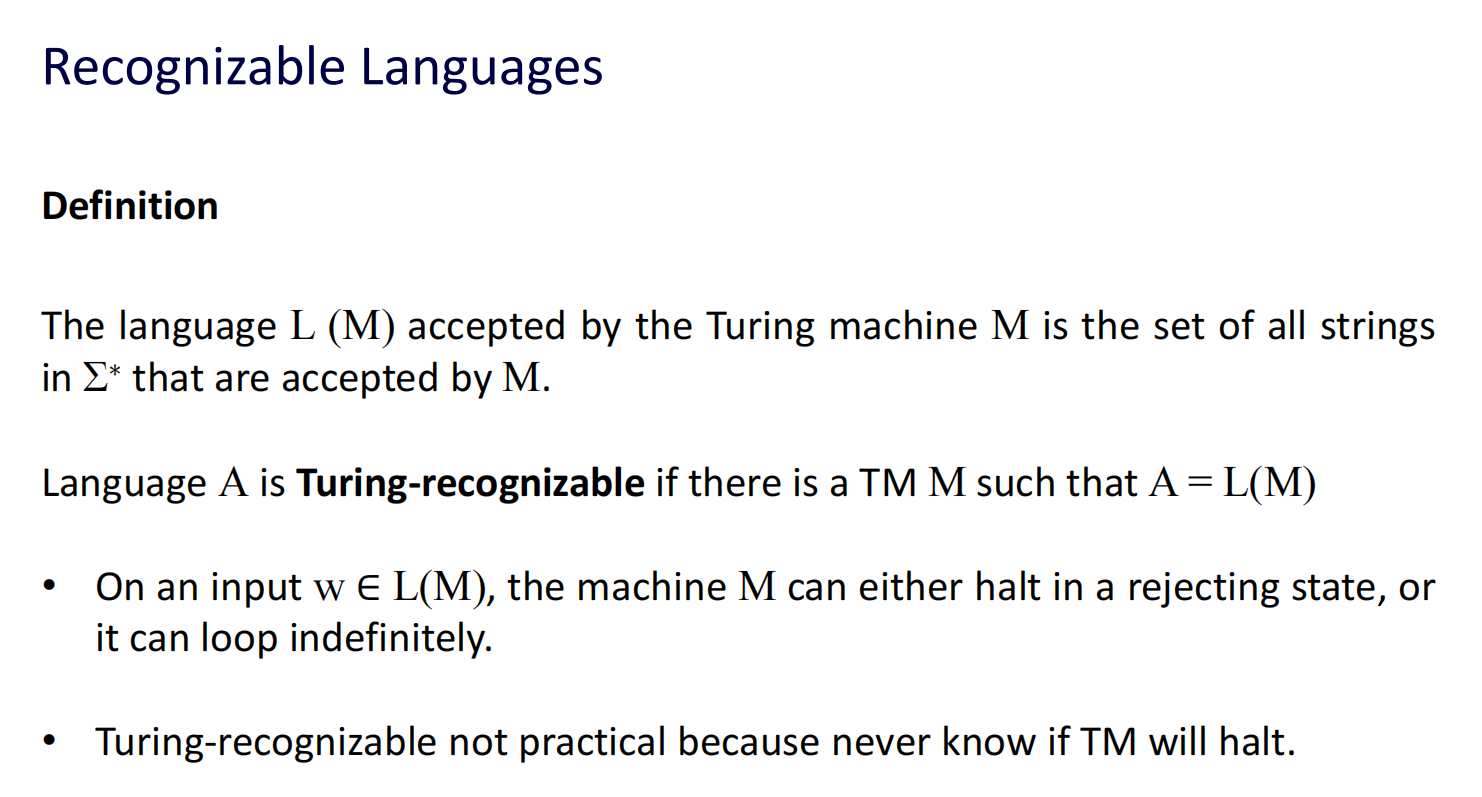
结果

- 最终配置 `C5` 是一个接受配置，形式为 `uqacceptv`，其中 `u = 101`，`v = □□□...`。

- 这表明图灵机 `M` 接受了输入字符串 `w = 101`。

Turing- recognization Languages & Decidibility

图灵可识别语言（Turing-recognizable Languages）的含义（lecture 9 p23）



定义

一个语言 `A` 被称为图灵可识别的，如果存在一个图灵机 `M` 使得 `A = L(M)`。也就是说，语言 `A` 中的所有字符串都是图灵机 `M` 所接受的字符串。

特点

1. 接受行为：

- 对于输入字符串 `w ∈ L(M)`，图灵机 `M` 可以在以下两种情况下停止：

- 停在拒绝状态（rejecting state）。

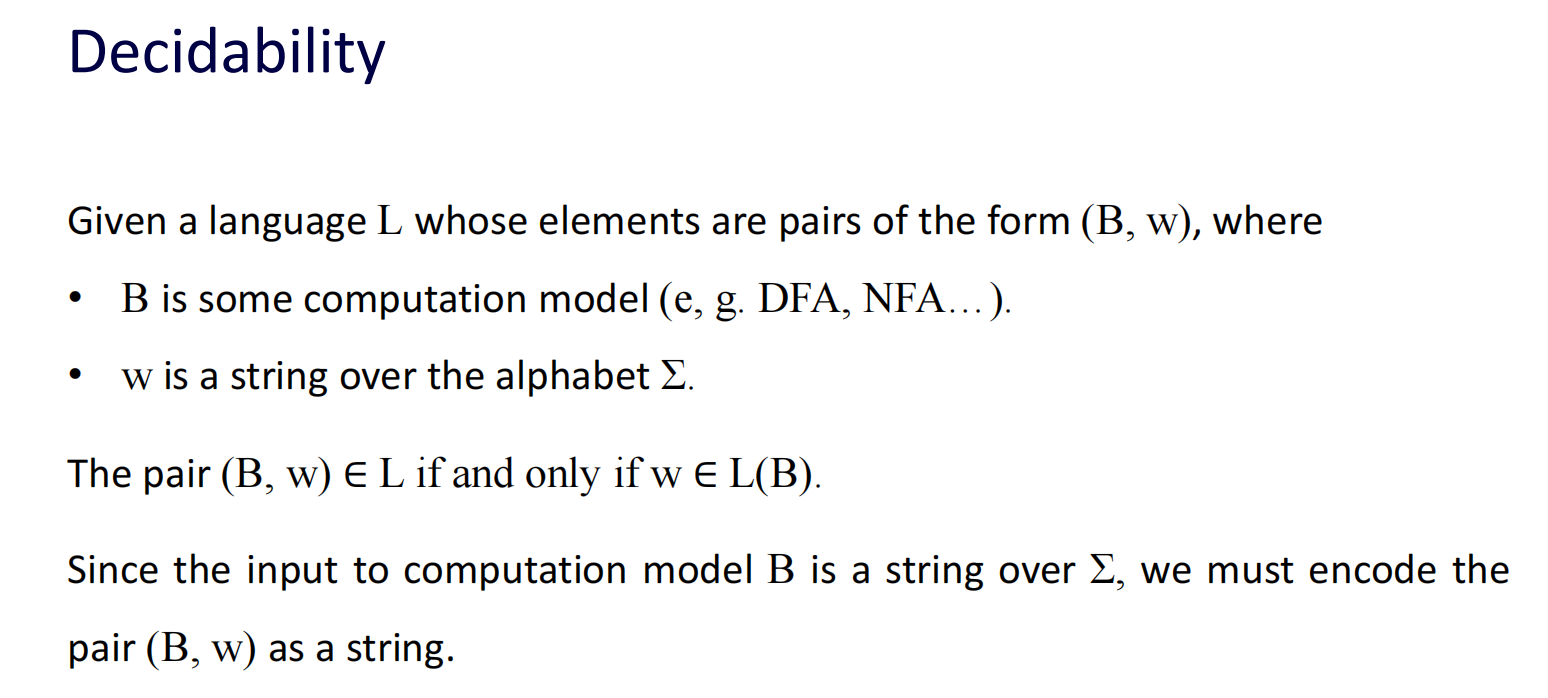
- 无限循环（loop indefinitely）。

2. 不确定性：

- 图灵可识别语言的一个重要特点是，我们无法确定图灵机 `M` 是否会在有限时间内停止。也就是说，即使 `w ∈ L(M)`，图灵机 `M` 也可能陷入无限循环。

实际意义

- 不实用性：由于无法确定图灵机是否会停止，图灵可识别语言在实际应用中并不实用。我们无法保证在有限时间内得到确定的结果。



可判定性（Decidability）

定义

可判定性涉及到一个语言 `L`，其元素是形如 `(B, w)` 的对，其中：

- `B` 是某种计算模型（例如 DFA、NFA 等）。

- `w` 是字母表 `Σ` 上的字符串。

`(B, w) ∈ L` 当且仅当 `w ∈ L(B)`，即字符串 `w` 属于计算模型 `B` 所接受的语言。

编码问题

- 由于计算模型 `B` 的输入是一个字符串，我们需要将 `(B, w)` 编码为一个字符串，以便图灵机能够处理。

**图灵可识别与可判定性的联系**

1. 基本关系：

- 可判定语言（Decidable Languages）：如果一个语言是可判定的，那么它一定是图灵可识别的。因为可判定语言对应的图灵机在输入字符串属于该语言时，总是会在有限时间内停止并接受；在输入字符串不属于该语言时，总是会在有限时间内停止并拒绝。

- 图灵可识别语言（Recognizable Language）：不一定是可判定的。因为图灵可识别语言对应的图灵机在输入字符串属于该语言时，可能会陷入无限循环。

2. 判定性问题：

- 对于一个语言 `L`，如果存在一个图灵机 `M` 能在有限时间内对任意输入 `(B, w)` 判定其是否属于 `L`，则称 `L` 是可判定的。

- 如果只存在一个图灵机 `M` 能识别 `L`，但无法保证在有限时间内停止，则称 `L` 是图灵可识别的。

举例说明

- 图灵可识别（recognizable）但不可判定的（undecidable）语言：

- 例如，停机问题（Halting Problem）是图灵可识别的，但不可判定的。存在一个图灵机可以识别哪些程序会停机，但无法在有限时间内对所有输入给出明确答案。

- 可判定语言：

- 例如，DFA 接受的语言是可判定的。给定一个 DFA 和一个字符串，可以在有限时间内确定该字符串是否被 DFA 接受。

联系：所有可判定语言都是图灵可识别的，但图灵可识别语言不一定是可判定的。

## Decidable language

### DFA is decidable

语言定义：`LDFA` 是由所有形如 `(B, w)` 的对组成的语言，其中 `B` 是一个确定性有限自动机（DFA），`w` 是 `B` 的输入字符串。

证明过程：

1. 检查编码：首先，检查输入 `(B, w)` 是否是“正确”的编码。如果输入不符合预期的编码格式，那么直接拒绝接受该输入。

2. 模拟DFA：如果输入是正确的，图灵机 `M` 将模拟DFA `B` 对字符串 `w` 的处理过程。具体包括以下步骤：

- `q`：DFA `B` 当前的状态。

- `i`：一个指针，表示在字符串 `w` 中的当前位置。

- 根据字符串 `w` 中当前位置的符号 `wi` 和DFA的转换函数 `δ(q, wi)`，更新状态 `q`。

3. 接受或拒绝：在模拟结束后，如果DFA `B` 最终停留在接受状态集合 `F` 中的一个状态 `q`，那么图灵机 `M` 接受输入 `(B, w)`；如果不在接受状态集合中，图灵机 `M` 拒绝输入。

由于DFA在任何输入下都将在有限步内终止在其状态集合中的一个状态，因此这个过程是可判定的。

### NFA is decidable

NFA语言的可判定性证明

语言定义：`LNFA` 是由所有形如 `(B, w)` 的对组成的语言，其中 `B` 是一个非确定性有限自动机（NFA），`w` 是 `B` 的输入字符串。

证明过程：

1. 检查编码：首先，和DFA的证明一样，检查输入 `(B, w)` 是否是“正确”的编码。如果不是，直接拒绝。

2. 转换NFA为DFA：如果输入是正确的，图灵机 `M` 将把NFA `B` 转换成一个等价的DFA `C`。这个过程通常涉及到构造DFA `C` 的状态集合，使得 `C` 的每个状态对应 `B` 中的一个状态集合。

3. 在DFA上运行：接着，图灵机 `M` 在DFA `C` 上运行，输入字符串为 `w`。这个过程实际上是对之前模拟DFA的步骤的复用。

4. 接受或拒绝：如果DFA `C` 接受字符串 `w`，那么图灵机 `M` 接受输入 `(B, w)`；如果 `C` 拒绝 `w`，那么图灵机 `M` 拒绝输入。

由于我们可以将NFA转换为DFA，并且DFA的语言是可判定的，因此NFA的语言也是可判定的。

总结来说，通过上述证明过程，我们可以确定DFA和NFA的语言都是可判定的。对于DFA，我们可以直接模拟其行为并作出接受或拒绝的决定；对于NFA，我们先将其转换为等价的DFA，然后在其上运行以作出决定。这两种方法都保证了在有限时间内能够给出明确的接受或拒绝的答案。

### CFGs are decidable

决策问题：给定一个CFG `G` 和一个字符串 `w`，判断 `G` 是否能生成字符串 `w`。

语言定义：`LCFG` 是由所有形如 `(G, w)` 的对组成的语言，其中 `G` 是一个CFG，`w` 是一个字符串。如果 `G` 能生成 `w`，即 `w ∈ L(G)`，那么 `(G, w)` 属于 `LCFG`；如果不能生成，那么 `(G, w)` 不属于 `LCFG`。

证明思路：

1. 转换至Chomsky正常形式：任何CFG都可以转换成Chomsky正常形式（CNF）。在CNF中，每条规则要么是 `A → BC` 的形式（其中 `A`, `B`, `C` 是变量，且 `B`, `C` 不是开始变量 `S`），要么是 `A → x` 的形式（其中 `x` 是终结符），要么是 `S → ε` 的形式（其中 `ε` 是空字符串）。

2. 生成过程的步数：在CNF中，对于任何非空的字符串 `w ∈ L(G)`，从开始符号 `S` 到 `w` 的推导过程恰好需要 `2|w| - 1` 步。每步推导要么是一个变量替换为两个变量的组合，要么是一个变量替换为一个终结符。推导的开始和结束各需要一步，中间的过程每增加一个终结符 `w` 的长度，就需要两步：一步是变量的增加，一步是变量的实现。

3. 枚举推导过程：由于规则集是有限的，我们可以尝试枚举所有可能的推导过程，构建解析树，并检查是否可以从开始符号 `S` 到达字符串 `w`。

### Emptiness of CFLs are decidable

CFL的可判定性

决策问题：给定一个CFG `G`，判断它是否能生成空语言，即 `L(G) = ∅`。

语言定义：`ECFG` 是由所有生成空语言的CFG `G` 组成的语言。

证明思路：

1. 规则集的有限性：CFG的规则集是有限的，这意味着我们可以尝试所有可能的规则组合来构建解析树。

2. 解析树的构建：我们可以尝试从所有可能的终结符列表开始，构建解析树，并检查是否能够通过应用CFG的规则到达开始符号 `S`。

3. 检查空语言：如果在尝试了所有可能的解析树构建之后，我们发现没有任何一条路径可以从开始符号 `S` 到达任何字符串，那么我们可以判断 `G` 生成的语言是空的。

总结来说，CFG和CFL是可判定的，因为我们有方法（如转换至Chomsky正常形式和枚举推导过程）来检查一个给定的CFG是否能够生成特定的字符串或者是否生成空语言。这些方法保证了在有限时间内能够得出明确的结论，因此CFG和CFL的可判定性得到了证明。

### The Language LTM is Turing-recognizable

语言定义：`LTM` 是由所有形如 `(M, w)` 的对组成的语言，其中 `M` 是一个图灵机，`w` 是一个字符串。如果图灵机 `M` 接受字符串 `w`，那么 `(M, w)` 属于 `LTM`；如果 `M` 不接受 `w`（无论是拒绝还是陷入无限循环），那么 `(M, w)` 不属于 `LTM`。

证明过程：

1. 通用图灵机：存在一个通用图灵机 `U`，它能够模拟任何给定的图灵机 `M`。也就是说，`U` 能够在输入 `(M, w)` 上执行 `M` 对字符串 `w` 的计算过程。

2. 模拟接受过程：如果图灵机 `M` 接受字符串 `w`，那么在通用图灵机 `U` 上模拟 `M` 对 `w` 的计算将会在有限步骤后停止，并且 `U` 将接受输入 `(M, w)`。

3. 模拟拒绝或循环过程：如果图灵机 `M` 不接受字符串 `w`（无论是拒绝还是陷入无限循环），那么在通用图灵机 `U` 上模拟 `M` 的行为将导致 `U` 要么拒绝输入 `(M, w)`，要么也陷入无限循环。

由于通用图灵机 `U` 能够识别出 `M` 是否接受 `w`，我们可以得出以下结论：

- 如果 `U` 在模拟过程中接受了 `(M, w)`，那么我们知道 `M` 接受 `w`，因此 `(M, w)` 属于 `LTM`。

- 如果 `U` 拒绝了 `(M, w)` 或者陷入了循环，那么我们知道 `M` 不接受 `w`，因此 `(M, w)` 不属于 `LTM`。

综上所述，由于存在一个图灵机 `U` 能够识别出 `LTM` 中的元素，`LTM` 是图灵可识别的。这意味着我们可以构造一个图灵机来判定任何给定的图灵机 `M` 是否接受任何给定的字符串 `w`，从而证明了 `LTM` 语言的可识别性。

### The Language LTM is undecidable

对角线方法（Diagonalization Method）

对角线方法的核心思想是通过构造一个特殊的元素，这个元素与给定的集合中的每一个元素都至少在一个属性上不同，从而揭示出集合的某种特性，通常是不可数性或不可判定性。这种方法的基本原理在于利用集合自身的结构来产生矛盾，以此来证明原假设的不成立。

以下是几个关键点，用以阐述对角线方法的核心思想：

1. 集合的表示：首先，假设我们有一个集合，这个集合中的元素可以以一种方式排列成一个序列或列表。

2. 特殊元素的构造：然后，我们构造一个新的元素，这个元素在某种程度上“模仿”列表中的每一个元素，但**同时确保它与列表中的每个元素至少在一个关键属性上不同**。这个属性通常是列表中元素的索引或它们在某个特定位置上的值。

3. 矛盾的揭示：通过这种构造，我们揭示出一个矛盾：如果新构造的元素属于原集合，那么它必须与列表中的某个元素相同，但由于它设计与列表中的每个元素都不同，因此它实际上不属于原集合。

4. 不可数性的证明：在证明集合不可数性的情况下，对角线方法展示了一个新的元素，它不在原列表中，从而表明原集合不能被完全列举出来，即它不可数。

5. 不可判定性的证明：在证明语言或问题的不可判定性时，对角线方法构造了一个特殊的语言或问题实例，这个实例在原假设下会导致一个矛盾，从而表明原假设（例如，存在一个判定器可以解决所有实例）是不成立的。

总的来说，对角线方法的核心思想是通过构造一个与所有现有元素都不同的新元素，从而揭示出集合的内在矛盾，进而证明集合的某种特性，如不可数性或不可判定性。

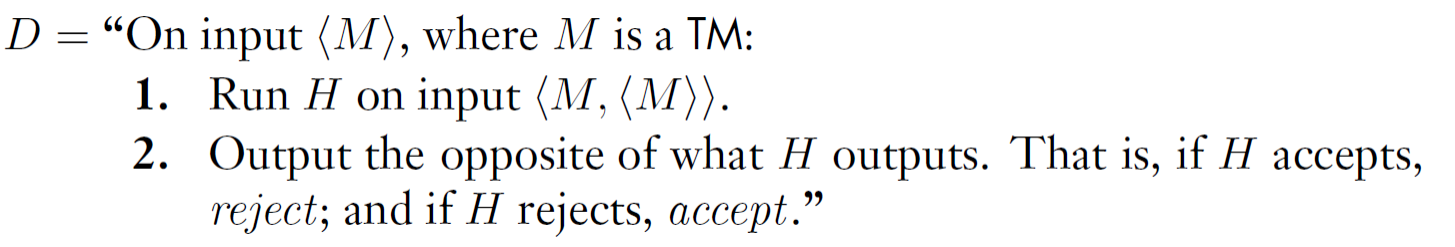
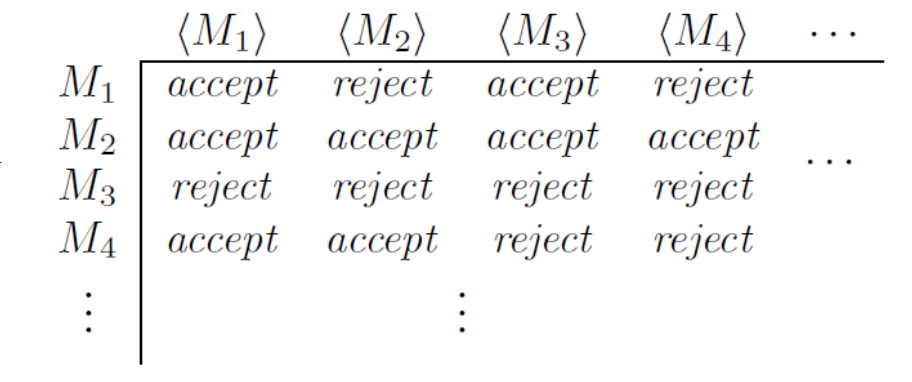
**图灵机语言不可判定的证明**

下面我们用对角线方法来证明图灵机语言 `LTM` 是不可判定的。

假设存在一个判定器 `H`，它能够决定任何图灵机 `M` 是否接受字符串 `<M>`（即 `M` 的描述）。

步骤1：列表表示

我们假设所有图灵机都可以用一个索引来表示，即 `M1, M2, M3, ...`。这是一个可数的列表，因为我们可以为每个图灵机分配一个唯一的索引。



步骤2：构造对角线元素

我们构造一个图灵机 `D`，它的行为是：对于任何输入 `<M>`，`D` 检查 `M` 是否接受 `<D>`。如果 `M` 接受 `<D>`，那么 `D` 拒绝 `<D>`；如果 `M` 拒绝 `<D>`，那么 `D` 接受 `<D>`。

步骤3：矛盾产生

现在我们来考虑 `D` 是否在 `H` 的列表中。如果 `D` 在列表中，那么根据 `D` 的构造，它将翻转列表中的对角线，这意味着 `D` 不在列表中。如果 `D` 不在列表中，那么根据 `H` 的定义，`H` 应该能够决定 `D` 是否接受 `<D>`，但这会导致 `D` 的行为与 `H` 的决定相矛盾。

矛盾分析

这个矛盾表明我们的假设是错误的，即不存在一个判定器 `H` 能够决定所有图灵机是否接受它们的描述。因此，图灵机语言 `LTM` 是不可判定的。

另一种证明方法

另一种证明方法是直接询问 `D` 是否接受 `<D>`。如果 `D` 接受 `<D>`，那么根据 `H` 的决定，`H` 应该接受 `<D, <D>>`，但这会导致 `D` 在步骤2中拒绝 `<D>`。如果 `D` 拒绝 `<D>`，那么 `H` 应该拒绝 `<D, <D>>`，但这会导致 `D` 在步骤2中接受 `<D>`。无论哪种情况，我们都会得到一个矛盾。

通过这两种方法，我们都证明了图灵机语言 `LTM` 是不可判定的。这表明我们不能构造一个图灵机来决定所有图灵机是否接受任何给定的字符串，这是图灵机理论中的一个基本结果。

### Instance of non Turing-recognizable languages

定理：一个语言 A 是可判定的当且仅当它是图灵可识别的并且它的补语言也是图灵可识别的。

证明：

- 必要性（只若部分）：这一部分比较简单。如果一个语言 `A` 是可判定的，那么存在一个判定器 `D`，它总是会在有限步骤内停止，并且会接受或拒绝输入。对于语言的补集 `A`，一个图灵机 `M` 接受输入当且仅当判定器 `D` 拒绝输入，反之亦然。因此，如果 `A` 是可判定的，那么 `A` 也是图灵可识别的，因为存在一个图灵机 `M1` 可以接受 `A` 中的所有字符串。同样，`A` 的补集也是图灵可识别的，因为存在一个图灵机 `M2` 可以接受 `A` 的补集中的所有字符串。

- 充分性（若部分）：现在，假设语言 `A` 和它的补集 `A` 都是图灵可识别的。我们有两个图灵机 `M1` 和 `M2`，分别识别 `A` 和 `A`。我们可以构造一个新的图灵机 `M`，它作为 `A` 的判定器。`M` 的行为如下：

- 当 `M1` 接受输入字符串时，`M` 也接受。

- 当 `M1` 拒绝输入字符串时，`M` 查询 `M2` 是否接受该字符串。如果 `M2` 接受，则 `M` 拒绝；如果 `M2` 拒绝，则 `M` 接受。

- 如果 `M1` 和 `M2` 都没有停止，`M` 可以选择任何一种行为（接受或拒绝），因为这种情况只会在 `A` 和 `A` 的补集都不包含输入字符串时发生，这种情况下 `A` 的真实值无关紧要。

通过这种方式，图灵机 `M` 可以在有限步骤内决定任何输入字符串是否属于语言 `A`，因此 `A` 是可判定的。

推论：`LTM` 不是图灵可识别的。

根据上述定理，如果 `LTM` 是可判定的，那么它必须是图灵可识别的并且它的补集也是图灵可识别的。然而，我们已经知道 `LTM` 是不可判定的（通过证明不存在一个可以决定所有图灵机是否接受其描述的判定器）。因此，根据定理的推论，`LTM` 不能是图灵可识别的。

这个推论表明，图灵机语言 `LTM` 的不可判定性意味着我们无法构造一个图灵机来识别所有图灵机是否接受它们的描述，这进一步表明 `LTM` 的补集也是不可识别的，从而符合定理的条件。