2026 年春季学期

1 实验背景与目标

在本实验中，你将亲手实现一个基于 **Gentzen 系统（G' 系统）**的命题逻辑自动定理证明器。该系统接收一个命题逻辑序贯 $\Gamma \vdash \Delta$，通过自动化的规则搜索，判断其是否可证。如果可证，输出对应的完整证明树；如果不可证，则输出一个使其为假的反例（此部分为 Bonus）。

2 实现语言与编程范式要求

你可以选择使用 **Coq**，或者常规的面向对象语言 **Java / C++** 来完成本实验。但如果你选择 Java 或 C++，**必须严格模拟 Coq 中的函数式编程特性**，包括而不限于：

- **代数数据类型 (Inductive Types) 的模拟**：你可以使用字符串来表示需要处理的序贯或者命题逻辑命题，但请注意对于字符串的处理需要使用递归函数进行，不能使用循环或者扁平化的字符串操作方法；
- **模式匹配 (Pattern Matching) 的模拟**：你可以定义不同的函数来实现对于不同结构的数据的处理，例如可以分别定义 `SolveAnd()` 和 `SolveOr()` 来处理最外层为 $\wedge$ 或 $\vee$ 的命题，但是请注意对于这些函数调用和管理。

3 系统设计与语法规范

我们在此提供系统的数学定义规范，你需要根据这些规范在代码中完成具体的数据结构定义。

3.1 输入字符串的文本规范

- **原子命题（变量）**：直接使用非负整数表示。例如，字符 `0` 代表变量 P_0，`1` 代表 P_1，以此类推。
- **逻辑连接词映射**：
  - 否定 ($\neg$)：使用波浪号 `~` 表示。例如：`~0` 表示 $\neg P_0$。
  - 合取 ($\wedge$)：使用和号 `&` 表示。例如：`0 & 1` 表示 $P_0 \wedge P_1$ 或 $P \wedge Q$。
  - 析取 ($\vee$)：使用竖线 `|` 表示。例如：`0 | 1` 表示 $P_0 \vee P_1$ 或 $P \vee Q$。
  - 蕴含 ($\rightarrow$)：使用由减号和大于号组成的字符串 `->` 表示。例如：`0 -> 1` 表示 $P_0 \rightarrow P_1$ 或 $P \rightarrow Q$。

- **结合律与括号**：使用英文圆括号（ 和 ）来显式声明运算优先级。任何包含二元连接词的复合公式，在嵌套时应当用括号包裹。
- **序贯分隔符**：
  - 序贯左侧的前件公式集 Γ 与右侧的后件公式集 Δ 之间使用字符串 `|-` 分隔。
  - 无论是左侧还是右侧，公式与公式之间一律使用英文逗号 `,` 进行分隔。
  - 可以在序贯中添加任意有穷个空格符。

语法示例对照表：

数学形态的序贯	对应的标准输入字符串
$P \vdash P$	`0 |- 0`
$P \to Q, P \vdash Q$	`(0 -> 1), 0 |- 1`
$\neg(P \vee Q) \vdash \neg P \wedge \neg Q$	`~(0 | 1) |- (~0 & ~1)`
$\vdash ((P \to Q) \to P) \to P$	`|- (((0 -> 1) -> 0) -> 0)`

3.2 证明树 (Proof Tree) 的规范

证明器在证明成功时，需要返回一个证明树数据结构。证明树的节点对应于 G' 系统中的推理规则。本次实验**不考虑 Cut 规则**，你只需要实现以下规则对应的树节点：

- **叶子节点**：`Ax` (公理规则)。
- **单分支节点**：`Not_L`, `Not_R`, `And_L`, `Or_R`, `Imply_R` (执行这些规则后，产生一个子目标序贯)。
- **双分支节点**：`And_R`, `Or_L`, `Imply_L` (执行这些规则后，产生两个独立的子目标序贯，必须两者均可证)。

3.3 核心工程机制：证明步数 (Step)

自动定理证明的核心是一个递归搜索函数。为了保证程序在任何情况下都能停机（尤其是满足 Coq 的严格终止性检查），你的自动证明器**需要**引入“步数 (Step)”机制。

- 你的求解函数需要接收一个额外的整数参数 `step`（如初始设为 100）。
- 每进行一次递归调用（应用一次 G' 规则），`step` 减 1。
- 当 `step == 0` 时，强制终止递归并返回“资源耗尽/无法判定”。

4 实验任务

Task 1: 定义核心数据结构和输入输出函数。使用你选择的语言，严格按照第 3 节的规范，定义出 `Prop`（命题公式）、`Sequent`（序贯）以及 `ProofTree`（证明树）的数据结构，以及接收字符串输入生成对应 `Sequent` 的输入函数，以及根据对应证 `ProofTree` 打印出结果的输出函数。证明树的输出形式没有限制。

Task 2: 实现 G' 系统证明器。编写求解程序，输入为给定序贯的字符串表示，输出为：

- 如果成功推导至公理，返回构造好的 `ProofTree` 并输出。
- 如果在设定的推理步数内无法完成推理或无可应用规则，则返回“无法证明”结果。

Task 3: (Bonus) 反例生成。当序贯不可证（在穷尽所有可逆规则后，叶子节点无法构成公理）时，G' 系统必然蕴含着一个反例。请扩展你的 `solve` 函数，使其在不可证时，输出一个**真值赋值 (Valuation)**。该赋值（例如一个 `Map<int, bool>`）必须使得 Γ 中的所有公式为 True，且 Δ 中的所有公式为 False。

提交方法：请将所有源程序代码打包后发送至课程邮箱。请注意我们的评分依据并不仅仅局限于程序的正确运行，还包括程序内部是否正确有效地模拟了命题逻辑以及 G' 系统推理所需的各类规则。

5 参考测试用例

Case 1: 基础公理测试 (Axiom Test)

- **数学序贯**：$P \vdash P$
- **标准输入字符串**：`0 |- 0`
- **预期结果**：`Provable` (可证)，返回单节点证明树。

Case 2: 分离规则推导 (Modus Ponens)

- **数学序贯**：$P \to Q, P \vdash Q$
- **标准输入字符串**：`(0 -> 1), 0 |- 1`
- **预期结果**：`Provable` (可证)。
- **证明树分支提示**：应用 `Imply_L` 后应拆分为 $P \vdash P, Q$ 与 $Q, P \vdash Q$ 两个子树，且两个叶子节点均触发 `Ax`。

Case 3: 德 · 摩根定律 (De Morgan's Law)

- **数学序贯**：$\neg(P \vee Q) \vdash \neg P \wedge \neg Q$
- **标准输入字符串**：`~(0 | 1) |- (~0 & ~1)`
- **预期结果**：`Provable` (可证)。

Case 4: 皮尔士定律 (Peirce's Law)

- **数学序贯**：$\vdash ((P \to Q) \to P) \to P$
- **标准输入字符串**：`|- (((0 -> 1) -> 0) -> 0)`
- **预期结果**：`Provable` (可证)。

Case 5: 反例生成测试 (Counter-example Test)

- **数学序贯**：$P \vee Q \vdash P \wedge Q$

- **标准输入字符串**：`(0 | 1) |- (0 & 1)`
- **预期结果**：`Unprovable` (不可证)。
- **预期反例赋值**：输出以下两种赋值之一均判定为正确：
  - `{0: true, 1: false}` （即 P 为真，Q 为假）
  - `{0: false, 1: true}` （即 P 为假，Q 为真）