arXiv:2603.29140v1 [cs.SE] 31 Mar 2026

使用 GPT 4.0* 根据需求设计有限状态机规格说明

Omer Nguena Timo[1],Paul-Alexis Rodriguez[2], 和 Florent Avellaneda[3]

[1] 魁北克大学乌塔韦校区，魁北克-加拿大
omer.nguena-timo@uqo.ca[2] 巴黎萨克雷大学，巴黎-法国 paul-alexis.rodriguez@universite-paris-saclay.fr[3]
蒙特利尔魁北克大学，魁北克-加拿大
florent.avellaneda@uqam.ca

摘要。 有限状态机（FSM）是反应式系统的可执行形式化规格说明。这些机器基于系统需求进行设计。需求通常记录在以自然语言撰写的文本文件中。有限状态机在模型驱动系统工程（MDE）的不同阶段发挥着至关重要的作用。例如，它们可用于自动化测试活动。有限状态机质量至关重要：有限状态机质量越低，在测试阶段未被发现的故障就越多，系统在生产环境中发生故障的风险也越高，这可能导致灾难性场景。因此，本文借鉴大语言模型领域的最新进展，提出一种基于大语言模型的框架，用于根据需求设计有限状态机。该框架还提出一种以专家为中心的方法，通过有限状态机变异和测试生成来修复大语言模型生成的有限状态机。本文还对大语言模型执行框架所述任务以及通过各种方法进行有限状态机修复的能力进行了实验分析与评估。本文使用模拟数据展示了实验结果。这些结果和方法为大语言模型提供了新的分析视角，有助于进一步发展机器学习技术及其在模型驱动系统工程中的应用。

关键词： 系统规格说明设计 · 有限状态机 · 需求 · 大语言模型 · 基于模型的测试 · 规格说明修复

1 引言

设计软件系统的形式化规格说明，有助于高效地分析系统、生成代码、测试、验证和维护系统。关于系统架构或行为的形式化规格说明，通常根据以自然语言表达的系统描述来设计。形式化规格说明通过有限状态自动机、米利机、UML类图等模型来描述。这项设计任务耗时较长，

* 由组织 x 资助。

如果完全由人类完成的话。自动化这项任务可以利用机器学习和大语言模型（LLM）领域的最新进展。

用于描述系统行为的状态转换模型已广泛应用于基于模型的系统工程（MDE）中。在这类模型中，触发转换的输入可以用标识符、变量上的复杂布尔表达式，或二者兼用来表示。前一种表示方式细节较少，适用于高层分析。米利机或有限状态机（FSM）是细节较少的模型示例，可用于研究系统的高层属性，并设计复杂的系统分析方法。更详细的模型则包含精确信息。详细模型的示例包括 UML状态图、符号自动机、Simulink Stateflow图、Verilog模型和程序。形式规格说明设计的部分自动化值得考虑，尤其是在完全自动化会归结为不可判定问题或产生近似规格说明时。部分自动化需要领域专门知识，由人类专家参与，以便在改进近似规格说明的过程中迅速作出重要决策。

机器学习已用于自动化规格说明设计及其他基于模型的系统工程（MDE）任务 [9]。随着大语言模型（LLM）的最新进展，机器学习如今正日益得到应用 [12, 1] 。大语言模型（LLM）是一种人工智能模型，旨在理解、生成和处理人类语言。部分大语言模型包括 OpenAI 的 GPT系列（如 GPT-3 和 GPT-4）、Meta 的 LLaMA系列、Google 的 BERT，以及 Microsoft 的 Turing-NLG。文献 [1]使用 GPT-4 生成 UML类图中属性和关联的对象约束语言约束，这些约束用于描述软件架构。这些约束可以是高质量 UML 图必须满足的属性。文献 [2, 8, 15]展示了大语言模型在生成系统行为规格说明、代码和测试方面的效用。具体而言，文献 [15] 使用大语言模型根据 Verilog规格说明生成 HDL程序。Verilog规格说明是状态转换机。为评估生成的 HDL程序的质量，文献 [15] 使用作者创建或从已知描述库借用的 Verilog规格说明[17]；他们还考虑了人工创建和自动生成的测试数据，用于测试生成的 HDL程序。文献 [15] 没有研究对生成的故障模型进行自动修复。当人工修复复杂且耗时时，就需要自动修复生成的模型。这种复杂性取决于生成模型中观察到的故障类型。基于区分测试和检查序列的方法已得到广泛研究，用于创建能够揭示有限状态机规格说明中故障的测试。据我们所知，这些方法尚未用于基于大语言模型的方法来设计规格说明。

本文使用模拟数据研究如何根据英文描述设计有限状态（米利）机。我们的研究旨在评估 GPT-4 使用米利机生成规格说明的准确性，并提出修复故障规格说明的方法。我们选择 GPT-4.0，是因为它提供了一个编程库，且该库被用于

其他工作。模拟数据能够代表真实数据，有助于开展我们的研究。我们的贡献如下：

– 我们生成模拟数据，用于评估所提出的方法。模拟数据包括随机生成的预言机有限状态机，以及针对这些预言机有限状态机自动生成的需求。– 我们设计大语言模型提示，根据有限状态机的文本描述生成有限状态机。– 我们提出四种方法，用于修复生成的故障有限状态机。第一种方法基于语法错误，其余方法则针对通过区分序列或检查序列检测到的错误行为。– 我们评估大语言模型根据文本描述生成正确有限状态机的能力，并评估专家在应用半自动修复方法时所需的工作量。

本文组织如下。首先，我们介绍米利机的一般概念以及本研究的工作流程。然后，我们介绍有限状态机模拟数据及其描述的生成、使用 GPT-4.0 生成规格说明，以及修复所生成规格说明的方法。接着，我们报告若干实验，以评估所提出的方法。最后，我们总结主要发现。

2 基于大语言模型的有限状态机设计方法

我们研究如何根据用英语撰写的描述自动设计确定性有限状态机（DFSM）。换言之，给定一段机器的文本描述，我们希望自动创建该机器的严格表示。确定性有限状态机是一种设备，它会针对来自环境的每个输入生成输出并改变其内部状态。它从已知的初始状态开始执行。

有限状态机（FSM）的严格表示是一个五元组 $\mathcal{S}=(S, s^0, X, Y, T)$，其中 S 是一个有限状态集，包含初始状态 s^0； X 和 Y 分别是有限、非空且互不相交的输入集和输出集； $T \subseteq S \times X \times Y \times S$ 是一个转移关系，元组 $(s, x, y, s') \in T$ 称为从 s 到 s'、输入为 x 且输出为 y 的一次转换。从状态 s 出发的转换集记为 $T(s)$。$T(s, x)$ 表示 $T(s)$ 中输入为 x 的转换集。对于转换 $t=(s, x, y, s')$，我们定义 $src(t)=s$, $inp(t)=x$, $out(t)=y$ 以及 $tgt(t)=s'$。对象 A 中的不确定转移集记为 $Unctn(A)$。如果 $|T(src(t)\ inp(t))|>1$，则转换 t 是不确定的，即从 $src(t)$ 出发的多个转换具有与 t 相同的输入；否则 t 是确定的。如果 $\mathcal{S}$ 没有不确定转移，我们称其为确定性有限状态机（DFSM）；否则称其为非确定性有限状态机（NFSM）。如果对于每个元组 $(s, x) \in S \times X$，都存在转换 $(s, x, y, s') \in T$，则 $\mathcal{S}$ 是完全指定的（输入完备有限状态机）。严格的描述可以使用严谨的语言编码，例如 DOT语言 [4]。图1展示了一个由4个状态和8次转换组成的DFSM。

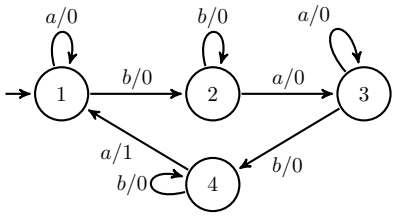

图1. 作为预言机的确定性有限状态机的图形化严格表示

清单 1.1. 图1中确定性有限状态机的描述

```
DFSM_描述:  = “当机器处于状态 s1 时，若输入 b，则返回 0，并转移到状态 s2。
```

在 s 中， $\mathcal{S}$ 的一次执行是由一系列有限转换组成的序列，这些转换在 $\mathcal{S}$ 的状态转换图中形成一条从 s 出发的路径，即对于每个 $i = 1...n-1$， $src(t_1) = s$, $src(t_{i+1}) = tgt(t_i)$。轨迹 x/y 是一对输入序列 x 和输出序列 y，二者长度相同。 e 的轨迹为 $inp(t_1)inp(t_2)\ldots inp(t_n)/out(t_1)out(t_2)\ldots out(t_n)$。 $\mathcal{S}$ 在 s 中的轨迹是 $\mathcal{S}$ 在 s 中执行所得的轨迹。令 $Tr_{\mathcal{S}}(s)$ 表示 $\mathcal{S}$ 在 s 中的所有轨迹集合。 $Tr_{\mathcal{S}}$ 表示 $\mathcal{S}$ 在初始状态 s^0 下的轨迹集合，它代表了 $\mathcal{S}$ 的行为。给定序列 $\beta \in (XY)^*$， β 的输入（分别为输出）投影记作 $\beta_{\downarrow X}$ （分别为 $\beta_{\downarrow Y}$），是通过从 β 中删除 Y（分别为 X）中的符号而得到的序列；若 β 是执行 e 的轨迹，则 $\beta_{\downarrow X} = inp(e)$（分别为 $\beta_{\downarrow Y} = out(e)$）称为 e 的输入（分别为输出）序列，并且我们称 $out(e)$ 是 $\mathcal{S}$ 在 s 中对输入序列 $inp(e)$（的应用）的响应。 $|X|$ 表示集合 X 的大小。

确定性有限状态机的文本描述由用于描述该确定性有限状态机的英语句子构成。描述可以包含关于状态和转移识别的结构信息，也可以包含将输入序列与相应输出序列关联起来的行为信息。一个确定性有限状态机可以对应多种描述。清单 1.1 展示了图1中确定性有限状态机预言机的一种描述。这些文本描述是人工合成的，并

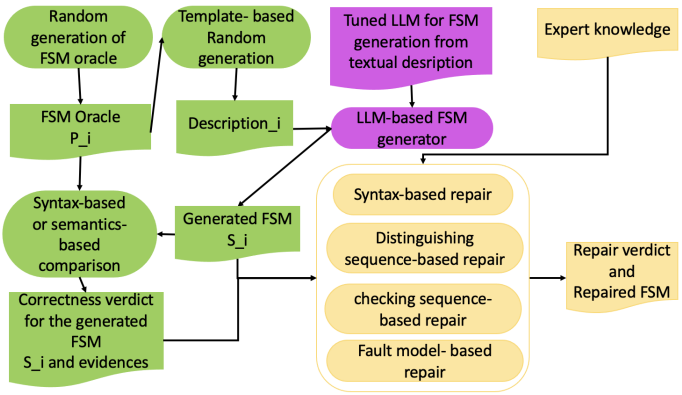

图 2。 基于大语言模型的方法：根据文本描述设计有限状态机

不对应于真实机器的描述。请注意，描述中并不总是在状态、输入或输出的标识符前加上关键词（状态、输入、输出）。由于缺少特定领域的上下文，大语言模型分析合成描述更具挑战性。

我们在图 2 中展示的根据描述设计确定性有限状态机的方法，利用了大语言模型（大语言模型）的能力。该方法包括对大语言模型进行提示工程，以根据文本描述生成确定性有限状态机。此外，还包括用于修复有缺陷的生成式确定性有限状态机的修复方法。这些方法依赖于人类专家提供的信息。只有扮演预言机角色的人类专家才能提供所需的信息。

3 基于 GPT 的确定性有限状态机生成

我们利用 GPT 大语言模型所提供的提示功能。提示是指通过设置参数并向大语言模型提供特定查询（提示），与其交互以生成所需答案（结果）。我们使用 GPT-4，因为它展现出了最具潜力的结果。

细化提示（也称为提示工程） [3, 16], 是一项需要工程师专业知识的手动工作。目前没有正式方法说明如何完成这项任务。通常会先进行微调，向大语言模型提供查询和预期答案，以帮助它们更好地理解需要完成的任务背景。还可以通过在发给大语言模型的消息中定义*系统角色* 来进行微调 。系统角色可以为大语言模型补充特定背景，明确我们希望完成的任务，并提供查询示例和预期结果。清单 1.2 展示了我们用于微调和查询 GPT-4o 的 Python 代码。消息中的*用户角色* 允许指定查询。消息中包含清单 1.1 中的文本描述。

清单 1.2. 提供给 GTP-4o 用于生成确定性有限状态机的提示

```
机器_角色 = " 你是一名专业的软件工程师，正在参与一个项目，该项目旨在根据




提示 = f"{ 确定性有限状态机_描述} 你能否按以下顺序创建前述
    自动机，并采用逗号分隔值格式：状态，
    输入， 输出， 下一个_状态， 状态应命名为 Si
    （其中 i 始终是一个数字），第一行应
    包含状态， 输入， 输出， 下一_状态， 而其他
    行中只能包含 Si 格式的状态名称（
    其中 i 始终为数字）、输入、输出以及
    Si 格式的下一状态名称（其中 i 始终为数字）
    ，各项信息之间不应有空格
    逗号。， here 例如：第一行：状态， 输入，
    输出， 下一_状态， 第二 行：S0，a，0，S2 第三 行：S1，b
    ,1 , S3 第四行：S2 ,c ,0 , S1 第五行：S3 ,d ,1 , S0 . 请
    请勿在 CSV 文件中添加任何注释。 请牢记 in
    机器应当完整且确定。"

响应 = 客户端 . chat . completions . create (
        model =" GPT -4 o" ,
        temperature =0.0 ,
        消息 = [
            {" role ": " 系统 " , "content": 机器_role },
            {" role ": "user" , " content ": 提示 }])
```

请注意，对于同一台机器的不同文本描述，大语言模型可能会生成语法和语义各异的确定性有限状态机。语法差异可能涉及状态名称、状态数量和转移。语义差异则涉及所生成的确定性有限状态机的输入输出序列集合。我们希望评估所生成确定性有限状态机的正确性，以及大语言模型完成生成任务的能力。这项评估颇具挑战，因为结果还取决于微调活动，而微调质量无法得到形式化保证。即使假设微调效果最佳，这一挑战依然存在。对于随机生成的确定性有限状态机，无需对模型进行微调。微调模型可以让大语言模型利用新提供的信息更好地完成所需任务；然而，在处理随机生成的确定性有限状态机时，并不存在能够提升大语言模型完成所需任务表现的模式或预期行为。尽管如此，微调提示至关重要，这也进一步说明了评估大语言模型性能的难度。在下一节中，我们将介绍评估方法及其相关的实证评估。

4 评估生成的确定性有限状态机的正确性

正确的生成式确定性有限状态机能够形式化地描述文本描述中预期的行为。只有预言机才能验证生成的确定性有限状态机是否正确。实际中，预言机由人类专家担任。然而，为了自动评估生成的确定性有限状态机的正确性，我们进行了一项仿真实验。在该实验中，我们使用充当预言机的确定性有限状态机来模拟人类专家。这类确定性有限状态机称为确定性有限状态机预言机。

4.1 使用预言机进行评估

评估方法是将预言机与生成的确定性有限状态机进行比较。我们考虑一种基于语法的比较方法，以及一种基于确定性有限状态机语义的比较方法。基于语法的比较结果是一组转移；基于语义的比较结果则是区分输入序列。这些结果体现了确定性有限状态机之间的差异。使用多个预言机进行实验，可以对大语言模型根据文本描述生成确定性有限状态机的能力进行统计评估。基于语法的确定性有限状态机比较会考察其状态和转移。当确定性有限状态机的状态之间存在一一对应关系时，才能进行此类比较。这种对应关系可以通过状态名称简单地建立，也可以通过确定性有限状态机之间的同构关系复杂地建立。对于我们使用合成数据开展的工作，简单的对应关系已经足够。因此，若满足以下条件，则两个确定性有限状态机在语法上存在差异：

– 它们的状态数量不同，或者使用的状态集合不同。– 它们使用相同的状态集合，且至少满足以下故障条件之一 [5] ：

- （故障类型 1：额外转移故障）：对于 M_2 中的某个 (s, x, y, t)，M_1 中不存在从 s 出发、输入为 x 的转换
- （故障类型 2：转移缺失故障）：对于 M_1 中的某个（s，x，y，t），M_2 中不存在从 s 出发、以 x 为输入的转移
- （故障类型 3：局部输出故障）在 M_1 中存在（s，x，y，t），且在 M_2 中存在（s，x，y'，t'），使得 $y \neq y'$
- （故障类型 4：转移故障）在 M_1 中存在（s，x，y，t），且在M_2 中存在（s，x，y'，t'），使得 $t \neq t'$

存在一种线性时间算法，可用于检查两个确定性有限状态机之间的语法差异。该算法会针对 $S_1 \times S_2$ 中的每一对状态，以及 X 中的每个输入，检查语法差异定义中的条件。此算法的结果是一组已识别故障，以转移集合 Δ 表示。Δ 中的每个转移都是存在故障条件的证据。假设两个确定性有限状态机之一充当预期的确定性有限状态机，则 Δ 可划分为两类转移：期望转移或非期望转移。**期望转移**在 $\Delta_d \subseteq \Delta$ 中出现于预期的确定性有限状态机中，但在其他确定性有限状态机中缺失。**非期望转移**在 $\Delta_u \subseteq \Delta$ 中缺失于预期的确定性有限状态机中，却出现在其他确定性有限状态机中。这些转移类别可用于修复生成的确定性有限状态机。

两个确定性有限状态机的基于语义的比较会考察它们的轨迹。直观而言，如果两个确定性有限状态机的轨迹集合不同，那么它们就不等价（即不同）。对于输入完备确定性有限状态机，如果某个输入序列在两个确定性有限状态机中对应不同的输出序列，就可以观察到这种差异。形式化地，给定输入序列 $x \in X^*$，令 $out_{\mathcal{S}}(s, x)$ 表示在状态 s 应用 x 时，$\mathcal{S}$ 可以产生的输出序列集合，也就是说 $out_{\mathcal{S}}(s, x) = \{\beta_{\downarrow Y} \mid \beta \in Tr_{\mathcal{S}}(s)$ 和 $\beta_{\downarrow X} = x\}$。给定有限状态机 $\mathcal{S}$ 的状态 s_1 和 s_2，以及输入序列 $x \in X^*$，如果 $out_{\mathcal{S}}(s_1, x) \neq out_{\mathcal{S}}(s_2, x)$，则称 s_1 和 s_2 是 x-可区分的，记作 $s_1 \not\simeq_x s_2$；此时，x 称为区分 s_1 和 s_2 的输入序列。如果 $out_{\mathcal{S}}(s_1, x) = out_{\mathcal{S}}(s_2, x)$，则称 s_1 和 s_2 是 x-等价的，记作 $s_1 \simeq_x s_2$。如果对某个输入序列 $x \in X^*$，s_1 和 s_2 是 x-可区分的，则称它们可区分，记作 $s_1 \not\simeq s_2$；否则，它们等价。令 $a \in X$。如果 x 不是区分 s_1 和 s_2 的序列，则称区分 s_1 和 s_2 的输入序列 $xa \in X^+$ 是最小的。若两个完整确定性有限状态机 $\mathcal{S}_1 = (S_1, s_1^0, X, Y, T_1)$ 和 $\mathcal{S}_2 = (S_2, s_2^0, X, Y, T_2)$ 具有相同的输入字母表和输出字母表，则当 $s_1^0 \not\simeq_x s_2^0$ 时，称它们可由输入序列 x 区分。

可以用区分自动机的概念，将对两个确定性有限状态机进行基于语义的比较形式化为一种线性时间算法。这种自动机模拟两个确定性有限状态机的并行执行，并识别出针对相同输入序列产生不同输出序列的并行执行。形式上，给定一个有限状态机 $\mathcal{S}_1 = (S_1, s_{1_0}, X, Y, T_1)$ 和一个有限状态机 $\mathcal{S}_2 = (S_2, s_{2_0}, X, Y, T_2)$，若有限自动机 $\mathcal{D} = (D \cup \{\nabla\}, d_0, G, \Theta, \nabla)$ 满足以下条件，则称其为 $\mathcal{S}_1$ 和 $\mathcal{S}_2$ 的**区分自动机**：其中 $D \subseteq S_1 \times S_2$，$\nabla$ 是一个接受（汇）状态，$\Theta \subseteq D \times X \times D$ 是转移关系

– $d_0 = (s_{1_0}, s_{2_0})$ 是 D– 中的初始状态，
对于任意 $(s_1, s_2) \in D$

- $((s_1,\ s_2)\ x,\ (s_1',\ s_2'))\in\Theta$，若存在 $(s_1,\ x,\ y,\ s_1')\in T_1$，$(s_2,\ x,\ y,\ s_2')\in T_2$
- $((s_1,\ s_2)\ x,\ \nabla)\in\Theta$，当且仅当存在 $(s_1,\ x,\ y_1,\ s_1')\in T_1$、$(s_2,\ x,\ y_2,\ s_2')\in T_2$和 $y_1\neq y_2$。

$\mathcal{D}$, $L_{\mathcal{D}}$ 的语言是为 $\mathcal{D}$从 d_0 到汇状态 ∇的执行添加标签的输入序列集合。我们可以证明， $\mathcal{S}_1$ 和 $\mathcal{S}_2$ 等价，当且仅当 $L_{\mathcal{D}}=\emptyset$；否则， $L_{\mathcal{D}}$ 中的输入序列可区分 $\mathcal{S}_1$ 和 $\mathcal{S}_1$。为 $\mathcal{S}_1$ 和 $\mathcal{S}_1$ 搜索区分输入序列，等同于在 $\mathcal{D}$中搜索从 d_0 到 ∇ 的路径；这可以通过线性时间算法完成。

4.2 实验结果

我们对所提出的方法进行了实证评估，该方法旨在根据自然语言描述生成确定性有限状态机。实验基于我们用 Python 编写的原型工具。该工具使用用于提示 GPT-4 的应用程序接口，并实现了针对确定性有限状态机的基于语法和基于语义的比较方法。评估采用随机生成的描述。

我们设计了一个 Python 模块，用于随机创建预期的确定性有限状态机，作为预言机，评估由大语言模型根据描述生成的确定性有限状态机是否正确。该模块的输入包括有限状态机的状态数，以及输入字母表和输出字母表的大小。图1中的确定性有限状态机由该模块创建。我们还设计了一个用于生成有限状态机描述的 Python 模块。有限状态机的描述由一组英文句子构成，有限状态机中的每个转换对应一个句子。我们设计了用于描述转换的英文句式模式。生成每条转换描述时，都会从可用模式集合中随机选择一种转换描述模式。每种转换描述模式对应于描述转换各组成部分（起始状态、目标状态、输入和输出）的一组模式。清单 1.1中的描述由已实现的模式生成。这些模式通过附录中清单 1.5 、清单 1.6 和清单 1.7 里的 toNL 函数实现。

GPT-4o 根据清单 1.1 中的描述生成了正确的确定性有限状态机。实证评估考察了具有 5、10 和 25 个状态的确定性有限状态机。这些确定性有限状态机定义了 5 个输入和 2 个输出。特定领域的有限状态机通常少于 25 个状态。输入和输出数量越多，大语言模型误读描述的风险就越高。对于给定的状态数，我们随机生成 30 个预言机（即预期的确定性有限状态机）。对于每个预言机，我们生成一段随机描述。对于每段描述，我们使用 GPT-4.0 生成一个确定性有限状态机。随后，我们将生成的确定性有限状态机与预言机进行比较。如果生成的确定性有限状态机与预言机不同，就称其存在故障。表 2、表 3 和表 4 展示了三项不同实验中生成的故障确定性有限状态机的数量。每项实验也用于评估一种修复方法。我们观察到，GPT-4o 最常出现的故障是输出故障和缺失转换故障。这一观察结果有助于制定用于修复生成的确定性有限状态机的故障模型。表 1 显示了以下故障类型在每台机器中的平均数量和最大数量：

状态数	类型 1	类型 2	类型 3	类型 4	全部
平均值					
5	0.0	0.03	0.0	0.0	**0.03**
10	0.07	0.43	0.27	0.33	**1.1**
最大值					
5	0	1	0	0	**1**
10	1	3	3	4	**11**

表 1. 生成的确定性有限状态机中每台机器的故障类型平均数和最大数

故障类型包括：类型 1（生成的机器中主要表现为非确定性）、类型 2（主要表现为输入上的转换缺失）、类型 3 和类型 4。对于每个状态数，我们使用 30 个预言机来生成结果。

5 修复生成的确定性有限状态机

本节探讨两个相互依赖的挑战。第一个挑战是确定性有限状态机修复本身，第二个挑战是评估大语言模型理解并应用修复查询（提示）的能力。以下小节将介绍四种应对这些挑战的方法。

5.1 基于 DFSM 预言机识别语法故障的方法

这种方法假设存在一个确定性有限状态机预言机，这虽然不切实际，但能够实现第二个目标。该方法的工作流程如下。设$\mathcal{P}$ 为一个确定性有限状态机预言机，π_{org} 为根据对 $\mathcal{P}$ 的描述构建、并发送给大语言模型的原始微调提示，大语言模型随后会返回生成的确定性有限状态机 $\mathcal{S}_0$。如果 $\mathcal{S}_0$ 在语法上与 $\mathcal{P}$不同，我们可以根据第 4.1 节中提到的错误，通过对 $\mathcal{P}$ 和 $\mathcal{S}_0$进行语法比较，构造一个包含语法错误和错误迁移的集合 $\Delta_0 = \Delta_{u_0} \cup \Delta_{d_0} = \{\tau_1, \tau_2, \cdots, \tau_n\}$。如果 Δ_0 非空，则生成的机器不正确。因此，我们开始修复该机器。为了修复机器，将 Δ_i 中的每个转移（首次比较 $\mathcal{P}$ 和 $\mathcal{S}_0$时使用Δ_0 ；第 i 次修复尝试中比较 $\mathcal{P}$ 和 $\mathcal{S}_i$ 时使用 Δ_i ）添加到由上一次修复尝试生成的提示 π_{i-1}中，从而生成增强提示 π_i。借助$\pi_0 = \pi_{org}$。为了生成 π_i，我们使用一个修复模板。每个期望转移 $\tau_d \in \Delta_{d_i}$ 都对应于在 $\mathcal{S}_i$中对 τ_d 执行的添加查询 π_{τ_d} ，其中 π_{τ_d} 是模板中针对转移 τ_d的期望转移所对应的句子。每个非期望转移 $\tau_u \in \Delta_{u_i}$ 都对应于从 $\mathcal{S}_i$ 中删除 τ_u 的删除查询 π_{τ_u} ，其中 π_{τ_u} 是模板中针对转移 τ_u的非期望转移所对应的句子。每个正确生成的转移 $\tau_c \in S_i$ 对应于

清单 1.3. 基于语法错误的修复查询模板

```
新_提示 =旧_提示
新_提示 += "更正自动机，使以下
    转移出现在生成的自动机中：\n"
对于元组在delta _u：
    新建_提示 +=tu.toNL() +"\n"
新建_提示 += "修正自动机，使得这些
    转换不会出现在生成的自动机中："
对于td在delta _d：
    新建_提示 +=td.toNL() +"\n"
新的_提示 += "这些转移是正确的，应该
    出现在生成的自动机中：\n"
用于 测试用例 在中:
    新的_提示 += 测试用例.toNL() +"\n"
新的_提示 += "请保持 此格式：状态, 输入, 输出,
    Next_状态 ，状态应命名为 Si（其中 i 是
    始终为数字），第一行应包含状态,
    输入, 输出, 下一_状态, 而其他行应仅
    包含 Si 格式的状态名称（其中 i 始终是一个
    数字）、输入、输出以及下一状态名称
    Si 格式（其中 i 始终是数字），不应该有
    各项信息之间只能用逗号分隔，不能有空格。这里有一个
    示例：第一行：状态, 输入, 输出, 下一个_状态,
    第二行：S0 ,a,0,S2 第三行：S1 ,b,1,S3 第四行：S2
    ，c，0，S1 第五 行：S3，d，1，S0。 不要 add any 评论 \n"
返回 新_提示
```

一个保留查询π_{τ_c}，其中π_{τ_c} 是模板中针对转移 τ_c的正确生成转移所对应的句子。我们将 π_{Δ_i} ={$\pi_{\tau_{u_1}},\cdots,\pi_{\tau_{u_n}},\pi_{\tau_{d_1}},\cdots,\pi_{\tau_{d_n}},\pi_{\tau_{c_1}},\cdots,\pi_{\tau_{c_n}}$}。 π_{Δ_i} 中每个成员的串联结果称为π_f，其中$\pi_f = \pi_{\tau_{u_1}}\ldots\pi_{\tau_{u_n}}\pi_{\tau_{d_1}}\ldots\pi_{\tau_{d_n}}\pi_{\tau_{c_1}}\ldots\pi_{\tau_{c_n}}$。 π_f是将模板应用于集合 Δ_i后得到的结果。随后，我们将 π_{i-1} 和π_f串联起来，生成第 i 次修复尝试所用的改进提示 π_i 。然后，将此提示作为查询提供给大语言模型，由其生成确定性有限状态机 $\mathcal{S}_i$。我们重复这一过程，直到大语言模型生成的确定性有限状态机与预言机等价，或者达到最大修复尝试次数。对于每种方法，最大修复尝试次数按如下方式计算： $max\ iter = |S|*|A|$，其中 S_是预言机的状态集合，A 是预言机的输入集合。所有方法均采用相同的计算方式。清单 1.3 展示了基于语法的修复所用的提示。

表 2 展示了使用随机生成的 DFSM 预言机评估修复方法有效性的结果。对于每个故障生成的 DFSM，我们估算 Δ 的大小。对于 5 个状态的机器，2 台故障机器均在 1 次尝试后得到修复。对于 10 个状态的机器，7 台机器在 1 次尝试后得到修复，1 台机器在 2 次尝试后得到修复，另有 1 台机器在 3 次尝试后得到修复。所有机器均已修复。该修复方法之所以成功，是因为向大语言模型提供具体的

状态数观测者数cles	nb. of故障生成的DFSM	平均size of Δ	Max size of Δ	修复成功进行rate
5	30	2	0.033	1	100%
10	30	9	1.1	11	100%
25	1	0	0	0	100%

表 2。基于语法错误的修复方法的评估结果

要在 DFSM 中执行的语法更改，无需对 DFSM 的行为进行任何分析。

5.2 基于区分序列的方法

该方法基于输入输出轨迹，用于评估大语言模型处理基于输入输出轨迹的提示的能力。轨迹定义了确定性有限状态机的语义。一个重要的假设是，预期的确定性有限状态机或确定性有限状态机预言机可用，以便自动计算区分序列。

该方法与第 5.1 节所述方法的区别在于，修复查询是使用基于区分序列的模板构建的。除此之外，过程类似，并且保留了迭代过程。给定预言机 $\mathcal{P}$ 和生成的确定性有限状态机 $\mathcal{S}_i$，我们使用区分自动机来估计 $\mathcal{P}$ 和 $\mathcal{S}_i$的区分输入序列的子集 $Iseq = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$。如果两个机器等价，则 $Iseq$ 为空。我们通过在区分自动机中随机搜索接受路径来估计这些序列。也可以为此任务定义其他搜索策略。每个输入序列 σ_i 在 $\mathcal{P}$中对应一个输出序列 γ_i。对于轨迹 $\tau_i = \sigma_i/\gamma_i$，我们设计了一个修复查询 π_{τ_i}。π_{τ_i} 对应于模板中针对轨迹 τ_i的句子，其中我们为给定输入指定了 $\mathcal{P}$ 中的预期输出。该方法中的提示 π_i 与前一种方法的构建方式类似，不同之处在于将连接到 π_{i-1}的π_f 内容有所不同。在该方法中，π_f 对应于每个 π_{τ_i}的串联。π_i 的构建方式与前一种方法相同。然而，无法保证该方法会终止，并使大语言模型生成与区分序列一致的规格说明。大语言模型可能会在一个不包含预言机所预期规格说明的规格说明空间中循环。停止条件和迭代过程仍与之前相同。

清单 1.4 展示了修复查询的模板。

表 3 展示了使用随机生成的确定性有限状态机预言机时，修复方法有效性的评估结果。对于每个有缺陷的生成式确定性有限状态机，我们估算修复成功率，即正确的确定性有限状态机修复候选项数量与有缺陷的生成式确定性有限状态机数量之比（确定性有限状态机在一次尝试中得到修正）。结果略逊于

清单 1.4. 基于区分序列的修复查询模板

```
新_提示 = 旧_提示
新_提示 += " 修正自动机，使得此输入
    提供给自动机的序列：\n"
新_提示 += sigma_i +"\n"
新建_提示 +=" 生成此输出序列：\n"
新建_提示 += 伽马_i +"\n"
新建_提示 +="请保持此格式：状态, 输入,
    输出, 下一_状态, 状态应命名为 Si（
    其中 i 始终为一个数字），第一行应
    包含状态, 输入, 输出, 下一_状态, 以及
    其他行只能包含 Si 格式的状态名称
    格式（其中 i 始终是数字）、输入、
    输出以及 Si 格式的下一状态名称（其中 i
    始终是数字）, 之间不应有空格
    每项信息之间仅用逗号分隔, 以下是一个示例：
    第一行：状态, 输入, 输出, 下一_状态, 第二
    行：S0 ,a,0,S2 第三行：S1 ,b,1,S3 第四行：S2 ,c
    ，0，S1 第五 行：S3，d，1，S0。不要 add any 注释 \n"
返回 新_提示
```

状态数 预言机数。故障		生成 DFSM	修复 suc- 行率
5	30	1	100%
10	30	7	0%

表 3。基于区分输入-输出序列的修复方法的评估结果

先前的基于语法的方法。这可以用大语言模型所采用的机制来解释，具体而言就是注意力机制。在处理少量信息时，注意力机制会保留较高比例的每个输入词，以对短语中的每个词进行编码；但在处理大量信息时，该机制使用的信息比例会降低。检查序列通常比区分序列更长，并且会为确定性有限状态机的分支结构引入大量信息。

5.3 基于检查序列的方法

现实考量促使我们采用这种方法。在实践中，人类专家扮演预言机的角色，他们只能根据系统的文本描述推断其期望行为（输入输出轨迹）。他们并不了解预期的确定性有限状态机的完整结构（状态和转移）。即使如此，我们仍希望评估大语言模型生成的确定性有限状态机是否正确，并在其存在故障时加以修正。我们提出一种基于所谓的检查序列的方法。检查

算法1 基于检查序列的修复过程

要求： 确定性有限状态机 d 的描述
要求： n状态数的最大值
要求： K 修复尝试的最大次数。
构建 π 的查询 d
设置 $\pi_r = \pi$
当 $K > 0$ **时执行**
通过向 GPT-4.0 查询 π_r，生成一个确定性有限状态机 $\mathcal{S}$
如果 $\mathcal{S}$ 不具备输入完备性，则将其转换为输入完备确定性有限状态机
估算 $\mathcal{S}$ 的检查序列 σ/γ
向专家获取输入序列 σ 的预期输出 γ_{exp}
如果 $\gamma == \gamma_{exp}$ **则**
返回 "$\mathcal{S}$ 是预期的确定性有限状态机。"
else
构建修复查询 π_τ，用于 $\tau = \sigma/\gamma_{exp}$
设置 $\pi_r = \pi_r\pi_\tau$
设置 $K = K - 1$
结束如果
结束循环
返回"修复失败"

状态数预言机数。故障数		生成的 DFSM	修复 suc-成功率
5	30	5	40%

表4。 基于检查序列的修复方法的评估结果

序列是一个包含 n个状态的确定性有限状态机 S 的一条轨迹；除非另一个包含 n 个状态的确定性有限状态机与 S等价，否则该轨迹不可能是任何其他此类确定性有限状态机的轨迹。估算确定性有限状态机的检查序列并非易事。现有算法 [7, 13] 是NP完全的。在这种方法中，有两种选择：将预言机或实际专家用作专家。两种情况下，流程都与前一种方法相同，不同之处在于，我们使用单个检查序列，而不是使用多个区分序列；预期输出则由预言机或专家给出。停止条件是检查序列的输出与专家或预言机给出的输出相同。达到最大修复尝试次数时，修复过程也可以停止。结果表明，这种修复方法无法阻止重新生成已访问过的确定性有限状态机。使用检查序列修复生成的确定性有限状态机的优点是，人类专家只需分析一个序列即可构建修复查询。算法1展示了修复过程。

表4展示了修复有效性的评估结果：在生成的5台故障机器中，1台经过1次尝试后得到修复，1台经过3次尝试后得到修复，另有3台未能修复。表4中的结果针对包含五个状态的确定性有限状态机。这是因为计算检查序列十分耗时，即使对于状态数有限的机器也是如此

（例如8个、9个等）。结果表明，基于使用检查序列细化提示的确定性有限状态机修复可能效率不高，或者需要设计更高效的提示。

5.4 基于故障模型的方法

估算检查序列十分复杂，而且检查序列通常很长，这可能会妨碍前述方法的应用。关于 GPT-4 弱点的先验知识，使我们能够识别生成的规格说明中最常见的故障 [11]。这些知识可用于构建确定性有限状态机修复候选项的修复域，并制定一种高效方法，从该修复域中挖掘合适的候选项。

变异机 [6] 可用于表示修复域。变异机是一种非确定性有限状态机（有限状态机），其中包含（定义）大量确定性有限状态机规格说明。我们可以根据大语言模型可能在生成的确定性有限状态机中引入的故障类型来构建它。每种故障类型都会在生成的确定性有限状态机中添加新的转移。这些新转移对应于用于修复生成的确定性有限状态机的合理补丁。图 3 展示了针对清单 lst 中描述所生成的确定性有限状态机的变异机。该机器在状态 3 中为输入 b 和输入 a 添加补丁（转移），还在状态 4 中为输入 a添加一个补丁。修复过程旨在从修复域中挖掘出正确的确定性有限状态机，这等价于选择恰当的补丁。挖掘算法采用迭代方式。每次迭代都会确定故障域中两个确定性有限状态机 $\mathcal{S}_1$ 和 $\mathcal{S}_2$ 的区分序列，以及故障域中所有确定性有限状态机都可能产生的输出序列列表。算法会向专家询问预期的输出序列，然后利用该序列识别并移除故障域中不正确的确定性有限状态机。我们采用文献 [10] 中基于 SAT 的挖掘算法，并由确定性有限状态机预言机扮演人类专家的角色。该修复过程总会终止。如果正确的确定性有限状态机包含在修复域中，该过程便会输出修复后的确定性有限状态机；否则会发生修复失败。

表5展示了基于故障模型的修复方法的评估结果。为了证明整个设计方法对提示工程的鲁棒性，我们采用了新的提示设计。我们根据观察到的情况，假设 GPT-4.0 会引入输出故障和转移缺失故障，并据此从生成的确定性有限状态机中构建变异机。根据这些观察，我们构建了修复域。最后，在必要时，我们会为修复域添加特定转移，以使其包含预言机。这确保修复过程在我们的实验中总能收敛。值得注意的是，在实验过程中仅有少数几次添加了特定转移，这表明基于对大语言模型弱点的理解来自动构建修复域既可行又有效。在实践中，预言机往往不可用，因此无法添加特定转移。在这种情况下，修复过程会在修复域内生成一个确定性有限状态机，使其与专家针对输出查询提供的所有答案一致，前提是这样的确定性有限状态机

状态数	5	10
自动生成预言机数量	30	30
nb. of 故障 已生成 DFSM	15	26
修复成功率	100%	100%
输出查询的最大数量	30	89
输出查询的最大长度	4	5
nb. 修复 域扩展—— 充了特定的转换—— 换	4	12
新增项的特定最大值 转移	4	5

表5。基于故障模型的修复方法的评估结果

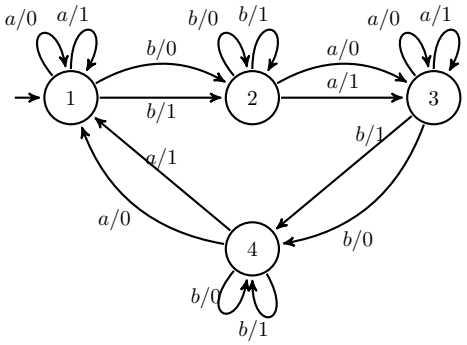

图3。变异机示意图，该机器定义了一个由256个确定性有限状态机构成的修复域，其中一个可能是预期的确定性有限状态机。该机器由图1中的确定性有限状态机构建而成，仅应用了对应于输出故障的变异操作。

存在。随后可以使用基于模型的测试技术验证生成的确定性有限状态机。

实验结果表明，专家在修复过程中可以回答数量可观的输出查询。需要采用方法来减少这一数量。一种可行的解决方案是考虑生成单个检查序列，以识别缩减后的修复域中的确定性有限状态机。我们预计此类检查序列会更短。

6 结论

我们提出了一种方法，使用大语言模型 GPT-4 根据文本需求设计反应式系统的形式化规格说明。该方法通过提示大语言模型根据描述生成确定性有限状态机规格说明，并修复生成的规格说明中的任何故障。我们使用随机生成的描述对该方法的精确度进行了实证评估。

结果表明，部分生成的规格说明掩盖了故障。
我们采用基于规格说明语法和语义的方法检测其中的故障。针对生成的规格说明，我们详细阐述了四种修复方法。首先，基于语法的方法虽然不切实际，却表明，在提示中加入已识别的语法故障来改进提示，可以提升大语言模型对先前错误的注意力，并近乎完全修复故障规格说明。相比之下，第一种基于语义的方法通过确定区分序列来增强提示，所得结果则大不相同。这种理论上可行但并不实际的方法需要更精准地理解和分析所指定的行为；它表明，几乎不可能通过改进提示，并加入大量表示确定性有限状态机预期及非预期行为的输入输出序列，来修复生成的规格说明。我们的实验表明，GPT-4 难以完成这种程度的分析，或者需要更先进的提示。”

第二种基于语义的方法是半自动的，也更切实可行、适用性更强。该方法利用自动构造的检查序列以及有关待设计系统的专家知识，确定唯一检查序列的预期输出。随后利用这些信息改进大语言模型的提示。该方法的结果略逊于第一种基于语义的方法，尤其是因为计算检查序列耗时较长。此外，单个较长的检查序列能够同时收集有关确定性有限状态机分支结构多个部分的信息。采用专门的注意力机制和更精细的提示，或许能进一步提升这种方法的有效性。

第三种基于语义的方法利用对大语言模型所引入故障的更深入理解。它使用修复域来限定生成的规格说明可能的修复空间，并通过与专家互动，确定与自动生成的相关输入序列相对应的输出序列，从而自动选择合适的修复方法。该方法不通过提示大语言模型来进行修复，而是依靠形式化方法，从已掌握的修复域中挖掘规格说明。这种方法前景可期，但需要更深入地理解大语言模型可能产生的故障，这将是未来工作的研究主题。

我们还观察到，随着机器规模增大，提供给大语言模型的信息量也随之增加，导致大语言模型产生的故障数量相应上升。大语言模型似乎缺乏进行精准分析的能力，也缺乏对关联（需要修正的输入输出序列和转移）进行“智能”理解的能力；在处理大量信息时，其精确度也会不足。尽管如此，在接近实际的规模下，大语言模型根据非形式化规格说明生成确定性有限状态机的表现良好；只需较少地介入大语言模型，就能提出切实可行的修复方法，例如基于故障模型的修复方法。
此外，未来的工作将考虑把这种方法应用于工业类系统的描述。根据我们在 [14]，中示例上的初步实验，我们预见工业类描述将添加特定领域的

信息，使大语言模型能够生成正确的规格说明，或使用根据输入输出序列构建的修复提示来修复生成的规格说明。

参考文献

1. Abukhalaf, S., Hamdaqa, M., Khomh, F.: Pathocl：基于路径的提示增强方法，用于借助 GPT-4 生成 OCL。载于：2024年IEEE/ACM首届人工智能基础模型与软件工程国际会议论文集。第 108–118 页（2024） 2. Bhandari, J., Knechtel, J., Narayanaswamy, R., Garg, S., Karri, R.: 面向有限状态机的大语言模型辅助测试平台生成与错误检测（2024），https://arxiv.org/abs/2406 17132 3 . . Denny, P., Kumar, V., Giacaman, N.: 与 Copilot 对话：探索使用自然语言解决计算机科学导论问题的提示工程。载于：第54届ACM计算机科学教育技术研讨会论文集，第 1 卷。第 1136–1142 页（2023） 4. Gansner, E., Koutsofios, E., North, S.: 使用 DOT 绘制图形（2006） 5. Ghedamsi, A., Bochmann, G., Dssouli, R.: 有限状态机的多重故障诊断。载于：IEEE INFOCOM'93 计算机通信会议论文集。第 782–791 页。IEEE（1993） 6. Koufareva, I., Petrenko, A., Yevtushenko, N.: 由用户定义故障模型驱动的测试生成。《通信系统测试：方法与应用》，第 215–233 页（1999） 7. Lee, D., Yannakakis, M.: 有限状态机测试的原理与方法综述。IEEE会议录 **84**(8), 1090–1123（1996） 8. Liu, M., Pinckney, N., Khailany, B., Ren, H.: VerilogEval：评估大语言模型的 Verilog 代码生成能力。载于：2023年IEEE/ACM计算机辅助设计国际会议（ICCAD）。第 1–8 页。IEEE（2023） 9. Mao, Y., Sahraoui, H.A., Lounis, H.: 使用机器学习技术验证可复用性假设：案例研究。载于：第13届IEEE自动化软件工程国际会议论文集（目录号 98EX239）。第 84–93 页。IEEE（1998） 10. Nguena Timo, O.: 挖掘由有限状态机建模的精确测试预言机。载于：IFIP国际软件测试与系统测试会议。第 20–36 页。Springer（2021） 11. Nikbin Azmoudeh, D.: 有限状态机变异算子分析。博士论文，卡尔顿大学（2024） 12. Oakes, B.J., Famelis, M., Sahraoui, H.: 构建领域专用机器学习工作流：实践现状的概念框架。ACM软件工程与方法学汇刊 **33**(4), 1–50（2024） 13. Petrenko, A., Avellaneda, F., Groz, R., Oriat, C.: 有限状态机推断与检查序列构造是同一事物的两个方面。《软件质量期刊》**27**, 651–674（2019） 14. Steffen, B., Howar, F., Merten, M.: 从实践角度介绍主动自动机学习。《永恒网络化软件系统形式化方法：第11届计算机、通信与软件系统设计形式化方法国际暑期学校（SFM 2011），意大利贝尔蒂诺罗，2011年6月13日至18日，进阶讲座》第 11 卷，第 256–296 页（2011） 15. Thakur, S., Ahmad, B., Pearce, H., Tan, B., Dolan-Gavitt, B., Karri, R., Garg, S.: Verigen：用于 Verilog 代码生成的大语言模型。ACM电子系统设计自动化汇刊**29**(3), 1–31（2024）

16. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., Schmidt, D.C.: 用于增强 ChatGPT 提示工程的提示模式目录。arXiv 预印本 arXiv:2302.11382（2023）
17. Wong, H.: 项目：about-hdlbits（2019），https://hdlbits.01xz.net/wiki/Project:About

附录

列表 1.5. 生成状态文本描述的模式

```
1 def toNL(self):2 描述候选 = [f" 状态 { self._label}",f"{self._label}"]3
rst =random.choice(descCandidate)4 returnrst
```

清单 1.6. 生成转换文本描述的模式 1 def toNL (self) -> str : 2 liste = [f"{ self . _ fromtoNL () } { self . _outputtoNL ()} and { self . _movetoNL () } { self . _inputtoNL () }" , 3 f"{ self . _outputtoNL () } and { self . _movetoNL ()} { self . _inputtoNL () } { self . _ fromtoNL () }"]4 rst =random . choice (liste) 5 return rst 6 7 def _systemtoNL (self) ->str : 8 liste = [" 它 " , " 系统 " , " 应用程序 "]9 rst =random . choice (liste) 10 return rst 11 12 def _fromtoNL (self) -> str : 13 liste = [f" 来自 { self . _src . t oNL () }" , 14 f" 来自状态 { self . _src . toNL () }" , 15 f" 处于状态 { self . _src . toNL () }" , 16 f" 位于 { self . _src . toNL () }" , 17 f" 当系统处于 { self . _src . toNL () } 时" , 18 f" 当它处于 { self . _src . toNL () } 时"]19 rst =random . choice (liste) 20 return rst 21 22 def _movetoNL (self) -> str : 23 liste = [f"{ self . _systemtoNL () } 转移到 { self . _tgt . toNL () }" , 24 f"{ self . _systemtoNL () } 到达 { self . _tgt . toNL () }" , 25 f"{ self . _tgt . toNL () } 被到达 "]26 rst =random . choice (liste) 27 return rst 28 29 def _ inputtoNL (self) -> str : 30 liste = [f" 如果输入是 { self . _input }" , 31 f" 如果输入 { self . _input } 出现 " , 32 f" 如果 { self . _input } 出现 " , 33 f" 当输入 { self . _input } 出现时" , 34 f" 当 { self . _input } 出现时"]35 rst =random . choice (liste)

36 return rst 37 38 def _outputtoNL (self) -> str : 39 liste = [f"{ self . _systemtoNL ()} 产生 { self . _output }" , 40 f"{ self . _systemtoNL ()} 返回 { self . _output }" , 41 f" , { self . _output } 被产生 " , 42 f" 输出 { self . _output } 被产生 " , 43 f" , { self . _output } 被返回 "]44 rst =random . choice (liste) 45 return rst

清单 1.7. 用于生成有限状态机文本描述的模式

```
def toNL(self) -> str :
  rst =""
  for cle in self._transitionsById.keys() :
     rst +=  " "+self._transitionsById[cle].toNL() + ".\n"
  return rst
```