arXiv:2512.15788v1 [cs.PL] 15 Dec 2025

结构化自然语言概率需求的自动形式化

Anastasia Mavridou[a, *], Marie Farrell[b, *], Gricel Vázquez[c, *], Tom Pressburger[d], Timothy E. Wang[e], Radu Calinescu[c], Michael Fisher[b]

[a]*KBR Inc.* 莫菲特机场 加利福尼亚州，加利福尼亚州 *94035*，美国[b]计算机科学系，曼彻斯特大学，牛津路，曼彻斯特，*M13 9PL*，英国[c]计算机科学系，约克大学，赫斯灵，约克，*YO10 5DD*，英国[d]*NASA* 艾姆斯研究中心，莫菲特机场，加利福尼亚州，加利福尼亚州 *94035*，美国[e]*RTX*技术研究中心，银巷*411*号，东哈特福德，*CT 06118*，美国

摘要

背景： 将自主和自适应行为集成到软件密集型系统中，对软件开发提出了重大挑战，因为环境或决策过程中的不确定性必须得到明确捕获。对于安全关键和任务关键系统，这些挑战更加突出，因为这类系统在设计和开发期间必须经过严格审查。其中一项关键挑战是如何指定使用概率构造来捕捉影响这些系统的不确定性的需求。为了进行形式化分析，此类需求必须使用精确的数学符号表示，例如概率逻辑。然而，期望开发人员直接使用复杂的形式化体系编写需求并不现实，而且极易出错。

目标： 我们扩展了美国国家航空航天局形式化需求 elicitation 工具（FRET）使用的结构化自然语言，使其支持对明确且正确的概率需求进行规

*通讯作者电子邮箱： anastasia.mavridou@nasa.gov (Anastasia Mavridou), marie.farrell@manchester.ac.uk (Marie Farrell), gricel.vazquez@york.ac.uk (Gricel Vázquez),tom.pressburger@nasa.gov (Tom Pressburger), timothy.wang@rtx.com (Timothy E.Wang), radu.calinescu@york.ac.uk (Radu Calinescu), michael.fisher@manchester.ac.uk (Michael Fisher)

定，并开发了一种自动化方法，将这些需求转换为逻辑公式。

方法： 我们提出并开发了一种形式化、可组合且自动化的方法，用于将结构化自然语言需求转换为概率时序逻辑中的公式。为提高对形式化结果的信任度，我们通过自动化验证框架和形式化证明，确保生成的公式格式良好并符合预期语义。

结果： 我们使用来自研究文献、公共代码库以及一项行业主导的案例研究中的 300 多项需求，开展了全面评估，以展示该方法的适用性。评估结果表明，我们的方法能够成功描述广泛的概率需求，证实了其表达能力和实用性。

结论： 扩展版 FRET 工具使开发人员能够用结构化自然语言描述概率需求，并自动将其转换为概率时序逻辑，从而使对自主系统和自适应系统的形式化分析更加实用且不易出错。

关键词：概率需求，需求形式化，概率时序逻辑，FRET，PCTL*

---

1. 引言

在核设施退役和太空探索等危险环境中，对自主系统的需求和期望日益增长 [1]。这些系统必须对通常存在不确定性的环境作出响应，同时依赖不完善且性能退化的传感器数据。这些领域中的自主系统通常属于（任务和／或安全）关键系统，因为系统故障可能导致代价高昂的操作失败，和／或严重威胁人身安全。对于关键系统，各种标准和法规（例如航空航天领域的 DO-178C）要求采用以需求为驱动的开发方法 [2]。自主性通常通过使用具备学习能力的组件（LEC）实现，再加上环境的多变性和感知的不完善，因此需要采用概率来有效刻画不确定性的需求 [3]。

需求工程（RE）面临两个重要挑战，而在自主系统中使用概率使这两个挑战更加严峻。首先，需求工程本质上是一项以人为中心的活动，因此，用于规范需求的语言对于需求的可用性和解释至关重要。因此，

开发人员通常使用直观的自然语言编写需求，但这可能会产生极大的歧义 [4]。基于逻辑的形式化语言等形式化语言能够避免这种歧义，但要求开发人员直接使用这些复杂的形式化体系编写需求并不现实，而且极易出错 [5]。加入概率会进一步加剧已有的歧义问题。第二个挑战与将需求连接到形式化分析工具有关，即验证开发中的系统是否满足其需求。针对以概率时序逻辑编写的规范，人们已经开发出多种分析技术，包括概率模型检测和运行时监控 [6, 7, 8, 9]。然而，要求开发人员直接使用如此复杂的形式化体系来规范需求，往往容易出错，甚至超出其技能范围。

为应对这些挑战，美国国家航空航天局艾姆斯研究中心在多个其他研究团队的贡献下，开发了开源工具 FRET [10], ，使开发者能够使用一种名为 FRETish 的结构化自然语言编写无歧义的需求。FRETish 需求的底层语义由该语言三个关键字段所赋值的组合定义：作用域、条件 和时序。这些字段值的每种独特组合都对应一个特定的模板键，它充当蓝图，指导如何将需求转换为形式化表示。FRET 提供 8 种作用域选项、2 种条件选项和 10 种时序选项，支持 $8 \times 2 \times 10 = 160$ 个不同的模板键。尽管经典 $FRET$ 已经具有很强的表达能力，但这些组合都无法支持对概率需求的描述。

具体而言，我们的贡献如下：

1. 扩展 FRETish 语言，包括：(i) 新增一个概率字段，用于表达概率需求；以及 (ii) 为条件 字段新增一个取值。这两项扩展为经典 FRETish 新增了 80 个模板键，并为其概率扩展新增了 320 个模板键，使总数大幅增加，从 160 个增至 560 个。

2. 将扩展 FRETish 语言形式化、组合式且自动地转换为概率计算树逻辑星号（PCTL*） [11, 12] 公式。我们证明生成的公式格式良好。

3. 基于预言机的自动化验证框架，有助于确保 e

生成的 PCTL* 公式符合需求的预期语义。

4. 通过对来自研究文献和工业界的多样化需求（包括形式化需求和自然语言需求）进行全面评估，检验我们方法的有效性。

我们的工具和评估需求均可在 [13] 获取。

1.1. 相关工作

需求的获取、形式化和分析一直是自动化软件工程领域的重要研究方向（例如， [14, 15, 16, 17]）。许多方法旨在通过定义可复用模式来简化形式化规约，然后将这些模式实例化以获得系统需求。一个著名的例子是规约模式系统（SPS） [18] ，其后续扩展涵盖了实时属性 [19]。Prospec [20], 、SPIDER [21] 和 SpeAR [22] 等工具可帮助用户编写基于模式的需求。EARS 方法 [23]提供了五种非形式化模板，已有研究表明，这些模板能够覆盖大多数高层需求。STIMULUS [24] 、ASSERT [25, 26], 和 SLEEC [27] 等其他工具则采用受限自然语言（例如拖放短语）来支持规约。然而，这些方法和工具均不支持概率需求。

其他方法侧重于为概率属性指定模式。ProProST [28] 定义了八种模式，这些模式源自相关文献，并以概率逻辑进行了形式化。与 FRET 一样，ProProST 支持使用结构化英语语法进行规范，以便自然地表达需求。Property Specification Pattern Framework（PSPFramework） [29], 整合并扩展了此前关于实时和概率属性模式的研究。PSPWizard 工具支持用户从其模式库中选择模式来构建形式化结果。类似地，QUARTET目录提供了 RPCTL（在 PCTL 中加入奖励 [30]）模式，用于规范精确的机器人任务需求 [31, 32], ；该目录还配有一个提供基于模式的领域特定语言的工具。

与其他依赖从预定义模式库中基于菜单选择的方案不同，FRET 允许用户直接使用受限形式的自然语言表达需求。这弥合了形式化规范（通常不够直观）与不受限自然语言（通常存在歧义）之间的差距。早期的经典 FRET

支持 160 个模板键；我们的扩展将其增加到 560 个，并支持表达不确定性。每个模板键都会以组合方式转换为公式，从而避免手动硬编码映射，提升可维护性。最后，我们提供了一个验证框架，用于系统地检查生成公式的正确性，提供了以往工作所不具备的保证。

本文其余部分安排如下。第2节概述我们的方法。随后，第3节介绍我们为支持概率需求的形式化而对 FRETish 规范语言所作的改进。第4节详细说明我们的组合算法，该算法可根据概率 FRETish 需求生成 PCTL* 规范。第5节介绍我们的自动化确认框架，旨在检查 FRET 生成的形式化结果是否正确。第6节对研究进行详细评估：我们整理了包含334条需求的语料库，并回答了三个与我们方法的可表达性和实用性相关的研究问题，同时评估该语料库中需求的共性。第7节讨论了我们在项目过程中遇到的若干挑战，以及如何将结构化自然语言用作 LLM 生成的自然语言与形式化需求之间的中介。第8节概述了适用于本研究的内部和外部效度威胁。最后，第9节简要总结全文，并展望未来的研究方向。

2. 概述

图 1 展示了我们框架的各个组件。该图分为两个主要部分：（i）左侧的预计算阶段 和（ii）右侧的用户交互阶段 。形式化过程由概率形式化器 在预计算阶段执行一次，然后以 JSON 文件的形式存储在概率缓存中。我们的概率公式生成算法 （详见第4.2节）会自动将每个模板键转换为 PCTL* 公式。为了支持生成 PCTL* 路径公式（其时序核心可以嵌入任意 LTL 表达式），我们使用时序逻辑结构化断言语言（SALT） [33]。SALT 支持将更高层级规范语言中的内容自动简化并生成线性时序逻辑（LTL） [34] 。所得

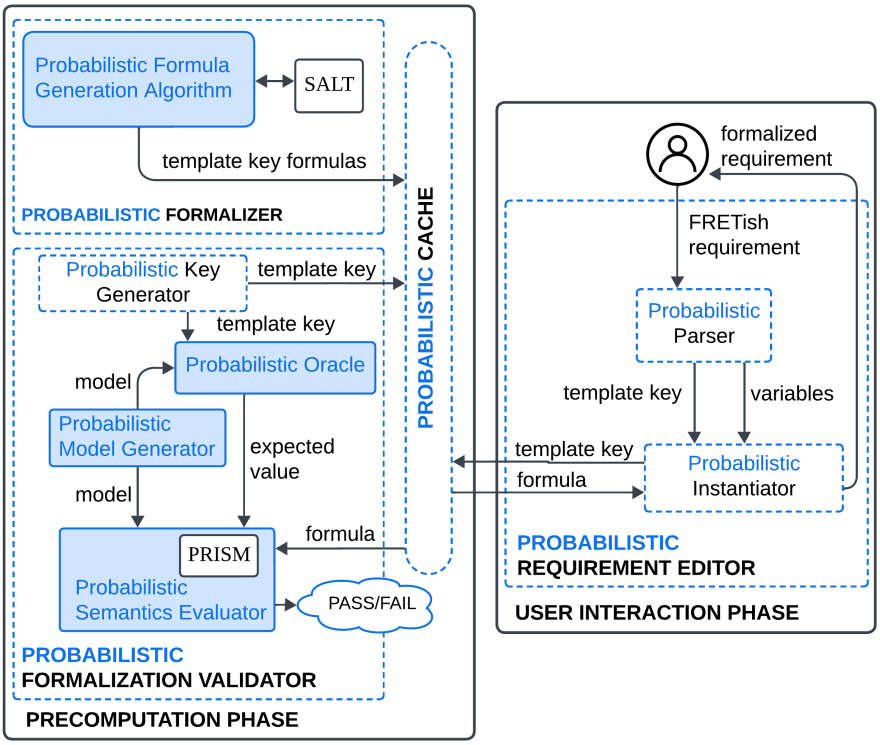

图 1：概率 FRET：对经典FRET组件的扩展以虚线表示；新增组件以实线和有色背景突出显示。

PCTL* 公式采用 PRISM 属性规范语言 [35]的格式表示。选择 PRISM 语言是因为它得到了各种概率分析工具的广泛支持，包括 PRISM [6], 、STORM [7] 和 iscasMC [8]模型检查器。此外，它还是一种便捷的中间表示，可轻松调整以兼容其他工具，例如 [36]。

为验证正确性，概率形式化验证器（第5节）提供了一个模块化框架，用于检查生成的公式是否准确表达预期语义。该框架包含扩展的概率键生成器，用于生成所有新的模板键，以及三个新组件：(i) 概率模型生成器，用于构造随机模型；(ii) 概率预言机，用于解释模板键并返回给定模型的预期结果；(iii) 概率语义评估器，它使用 PRISM 验证在模型上评估 PCTL* 公式后，结果是否与预言机的输出一致。

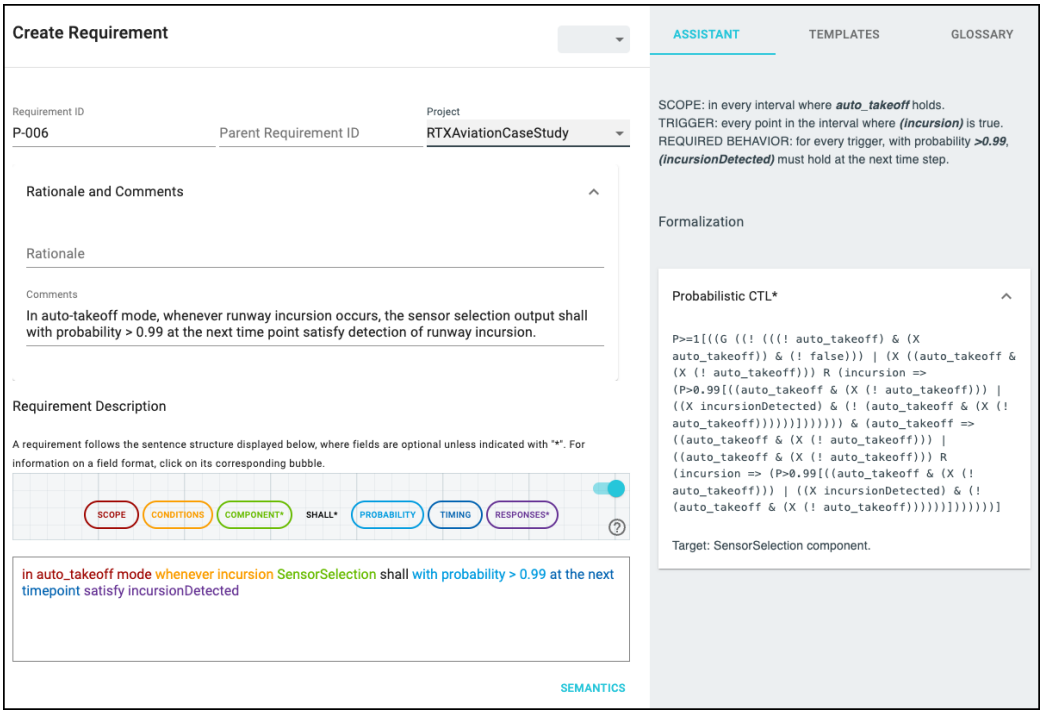

图 2：FRET 的概率需求编辑器。左侧以 FRETish 输入需求。需求最初未经结构化的自然语言版本（纯英文）输入在“评论”字段中。右侧，FRET 会解释需求的语义，并显示生成的 PCTL* 公式。

在图 1 的右侧，我们可以看到用户如何与 FRET 交互。需求使用扩展 FRETish（第3节）编写。

图 2 展示了扩展版 FRET 编辑器，其中正在输入需求 **[P-006]**。该需求源自 RTX[1] 技术研究中心（RTRC）的一项案例研究，研究内容聚焦于自主滑行、起飞和着陆系统（第 6.2 节），并贯穿本文作为示例。需求的自然语言版本输入在编辑器的 Comments 部分。用户输入时，概率解析器会识别相应的模板键并提取相关变量。随后，概率实例化器 从概率缓存中检索相应的 PCTL* 公式，并使用提取的变量对其进行实例化，

[1]RTX 是由柯林斯航空、普惠公司和雷神公司组成的企业集团。

然后将形式化后的需求返回给用户，如图 2 右侧的*assistanttab* 所示。

3. 需求规范语言

本节介绍我们对经典 FRETish 语言所做的扩展。为完整起见，由于我们的形式化算法（第4.2节）使用了完整的 FRETish语法，我们将描述经典 FRET 构造 [10, 37]以及我们所做的增强，并在适当之处明确区分二者。表1展示了扩展 FRETish 语法的语法，其中语法扩展以蓝色标出，而经典 FRET 构造则以黑色字体显示。该语法的格式遵循 ANTLR 的语法语法，而 ANTLR 的语法语法基于扩展巴科斯-瑙尔范式（EBNF）。在语法中，终结符以大写字母和**粗体显示，非终结符则以等宽字体显示。可选元素用'( )?'表示。例如，（`WHEN`）? `IN` 表示，如果省略 `WHEN`能使句子更符合用户的自然语言表达，则可以省略它。符号'()*'表示分组元素重复零次或多次，而''表示可选方案**。

通过我们的扩展，一个FRETish 需求由七个字段组成。输入需求时，FRET 编辑器会动态为对应各字段的文本着色（见图 2）。这些字段是作用域；条件；组件*；`shall*`；概率；时序；以及响应*。带星号的字段为必填项。

我们使用需求 **[P-006]** 来说明FRETish字段。我们先介绍必填字段。组件字段指定该需求适用的系统组件（例如，传感器选择），并定义为字符串。`shall` 关键字表示组件的行为必须符合该需求。响应字段的形式为满足 R（例如，满足 检测到侵入），其中R 是一个布尔表达式，组件的行为必须满足该表达式。

作用域字段指定与组件行为相关的运行模式（例如，处于自动_起飞模式），其定义了需求受约束的时间区间。若省略，作用域默认为全局，即整个执行过程。对于某个模式（例如，自动_起飞），FRET支持七种关系：之前模式（需求仅在模式首次成立之前严格生效）；之后模式（需求仅在模式最后一次成立之后严格生效）；处于模式（或

表1：扩展 FRETish 语法。标识符是一个字母数字字符串，概率_界限$\in \mathbb{R}^{[0,1]}$ 和数字 $\in \mathbb{R}^{+}$。语法扩展以蓝色突出显示，而经典FRET 构造则以黑色字体显示。

Rule		定义
需求	::=	（作用域）？（条件）？ （组件**shall** | **shall** 组件） **(概率)?** (时序)? 响应
作用域	::=	**仅**（（**在..** （（ 作用域_模式 | **当.....时** 作用域_条件 | （**之后** | **之前**）（作用域_模式 | 作 | **除非** （（**期** | 作用域_模式 | **当** 作用域_条件） | **非**? **在** 作用域_模式 | （**在** | 用域_模式 | **除非在** 作用域_模式 | **当.....时** 作用域_条件 (**之后** | **之前**) (作用域_模式 | 作用域_条件)
作用域_模式	::=	(**模式** 模式_名称) | (模式_名称**模式**) | 模式_名称
mode_name	::=	ID
作用域_条件	::=	布尔值_表达式
条件	::=	(**与**)? 条件_表达式
条件_表达式	::=	限定_条件1 (限定_条件2)*
限定_条件1	::=	限定词_词前缀_条件（**是**（**真** | **假**））?
限定_条件2	::=	(**且** |**或**)? 限定词_词 前置_条件 （**是**（**真** | **假**））?
限定词_词	::=	**当** | **非** | **其中** | **如果**
前置_条件	::=	布尔值_表达式
组件	::=	(**该**)? 组件_名称
组件_名称	::=	ID
概率	::=	**具备** 概率_辅助项
概率_辅助项	::=	**概率** 关系_运算符 概率_界限
关系_运算符	::=	**<** | **<=** | **>** | **>=**
时序	::=	**在.....以内** 持续时间 | **持续**持续时间 |**之后** 持续时间 |**直到**停止_条件 | **之前**停止_条件 |**在**（**第一个** | **下一步**）**时间点** |**立即** | **最初** | **从不**
停止_条件	::=	布尔值_表达式
持续时间	::=	数字 时间单位
时间单位	::=	**时钟周期** | **微秒** | **毫秒** | **秒** | **分钟** | **小时**
响应	::=	满足
满足	::=	**满足** 后置_条件
后置_条件	::=	布尔值_表达式

同义词包括 期间、处于……时、处于……时、当……时）：当组件处于某个模式时，需求会得到执行；而不处于 该模式（或同义词除……之外、除非处于……）表示当组件不处于该模式时，需求会得到执行。有时，需要指定需求仅在特定时间区间内得到执行，这意味着在该区间之外不应满足该需求。为此，提供了仅在……之后、仅在……之前和仅在……期间这些作用域。

该条件字段是一个布尔表达式，用于在指定作用域内触发响应。我们的扩展支持两种类型：(i) 正规条件，通过*upon* 关键字（或同义词当、其中和如果）触发响应，使其在条件变为真（由假变真）时发生；(ii) 一种新的保持条件，通过*whenever* 关键字（例如whenever incursion）触发响应，使其在条件为真的所有时间点发生。

本文引入的新概率字段用于指定时序响应的概率下界或上界，表示系统在指定限制内满足响应时序约束的可能性（例如，以概率 > 0.99）。

该时序字段指定相对于下一时间点等情况，响应应在何时成立作用域和条件。FRET 支持十种时序选项：立即（同义表达包括最初、在同一时间点、在第一个时间点）、下一步、从不、最终、始终、在……以内n 个时间单位、持续n 个时间单位、在n 个时间单位之后（即在N 个时间单位内不成立，并在第 n+1st 个时间单位成立）、直到（响应持续成立，直到出现停止条件），以及之前（响应在出现停止条件之前成立）。省略时序时，默认使用最终。

总之，这项工作的关键扩展包括新增的概率字段，使概率需求得以表达；以及引入保持条件。后者显著增强了经典FRETish的表达能力（新增了 80 个模板键），并支持表达无限多次成立（G F）的需求，而这类需求此前在FRET中尚不支持。它还使某些概率模式得以表达，例如概率响应 [28]。为了让需求表达更加自然，我们还扩展了FRETish 语法，为作用域、条件和时序字段中的语义等价构造提供了多种句法替代形式。例如，*in* 模式构造现在可以写作**DURING** 作用域、**WHEN IN** 作用域或**WHILE** 作用域_条件。

3.1. 概率字段的位置。

概率字段的位置是根据对工业案例研究和学术文献中的概率需求所做的分析确定的。我们考虑了几个候选位置，包括将其放在必需的响应字段之前，以及放在需求开头、作用域字段之前。

将概率字段放在响应字段之前，会产生一个由 PCTL* P 算子包裹的布尔表达式，其公式的计算结果为 0 或 1。这种构造限制了表达能力，也无法支持对有意义的概率约束进行规范。

将概率字段放在FRETish结构的最开头，即作用域之前，会将整个时序需求置于概率算子之下。虽然这种位置支持某些类型的概率属性，但它限制了表达更细致公式的能力。具体而言，它无法规范概率约束仅适用于条件语句后件的需求（例如："如果 A，那么以 ≥ 0.9 的概率最终满足 B"），也就是说，概率约束仅适用于需求中的时序-响应部分，而非整个时序表达式。此外，这种位置还会带来空泛满足的风险：如果条件很少成立，即使响应始终失败，整个需求仍可能以很高的概率空泛满足。

因此，我们决定将概率字段直接放在时序字段之前。这种位置安排在包括 PCTL 在内的概率逻辑中很常见*，并支持对各种有意义的概率属性进行规范。它还提供了灵活性——当省略作用域和条件时，概率适用于整个需求。最后，使用作用域和条件时，我们避免了上述空泛性问题，因为概率仅约束需求中的时序-响应部分。

4. 组合式形式化

我们的形式化算法（算法1）采用组合式方法，也就是说，我们不是为每个模板键手动定义单独的公式，而是根据模板字段的值，通过组合子公式系统地构造完整的 PCTL* 公式。例如，考虑需求 **[P-006]** 及其对应模板键的构造。

如引言所述，经典 FRET 模板键由以下字段的值导出：作用域、条件和时序字段。通过扩展，我们在模板键定义中引入第四个组件，即概率字段。因此， **[P-006]** 的模板键为： $[in, holding, bound, next]$。为生成相应的 PCTL* 公式，算法会对 in 作用域、$holding$ 条件、概率界限和 $next$ 时序的语义进行解析并组合。

我们的算法建立在 Giannakopoulou等人 [39, 37],先前工作的基础上。该工作提出了一种组合算法，用于根据经典 FRETish需求（即表1中未高亮显示的语法部分）生成 SALT公式，并将其转换为 LTL 公式。我们的方法复用了 [37]中介绍的 FRETish作用域、时序和正规条件的语义。为保持一致，在可能的情况下，我们复用了 [37] 中的术语和变量名称。接下来，我们将简要讨论 PCTL*、相关的 SALT构造以及 FRETish作用域区间。

4.1. 预备知识

4.1.1. 概率计算树逻辑星号

PCTL* [11, 12] 是一种概率逻辑，它源自分支树逻辑 CTL* [40] ，通过使用概率算子 P替换路径量词 $\exists$ 和 $\forall$而得到。该概率算子定义了系统演化概率的上界或下界。例如，公式 $P_{\geq p}(\phi)$在给定时间为真，当且仅当系统未来演化满足 ϕ 的概率至少为 p [9]。

接下来，我们简要介绍 PCTL*。关于该逻辑的详细介绍，请参阅 [11, 12]。对于一个有限的原子命题集合 AP ，$PCTL^*$ 状态公式 Φ 和 $PCTL^*$ 路径公式 Ψ 由以下语法定义：

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid P_{\sim p}[\Psi]$$
$$\Psi ::= \Phi \mid X\Psi \mid \Psi_1 U \Psi_2 \mid \Psi_1 U^{\leq n} \Psi_2 \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi$$

其中， $a \in AP$、 $\sim \in \{<, \leq, >, \geq\}$、 $p \in [0,1]$和 n均为自然数。X、 U、 $U^{\leq n}$分别是时序算子“下一步”、“直到”和“有界直到”。 X 算子指下一个时间点，即当且仅当Ψ 在下一个时间点为真时， $X\ \Psi$ 才成立。时间有界的“直到”公式 $\Psi_1 U^{\leq n} \Psi_2$要求 Ψ_1 在时间步 $[0, x)$ 内始终成立，并且 $x \leq n$ 和Ψ_2 在时间点 x变为真。将 n设为 ∞，即可得到无时限路径公式 $\Psi_1 U \Psi_2$ 。其他布尔算子，例如析取（$\vee$）和蕴含（$\Rightarrow$），可以用基本布尔算子表示：

否定（¬）和合取（∧）。对于路径公式，时序算子“最终” F 和“全局” G（及其有界变体 $F^{\leq k}$和 $G^{\leq n}$）可以由“直到”算子 [9]推导得出；当且仅当 ϕ 在当前或某个未来时间点成立时， $F\ \phi$ 才为真；当且仅当 ϕ在未来始终为真时， $G\ \phi$ 才为真。请注意， Ψ 可以是任意 LTL 表达式，因此可以对路径公式进行任意组合。

4.1.2. 时序逻辑结构化断言语言（*SALT*）

我们使用 SALT 生成 PCTL* 路径公式，利用其命题算子：非，与，或，蕴含，时序算子：直到，始终，最终和下一步，以及定时修饰符：定时[∼]，其中 $\sim$ 是 $<$ 或 $\leq$，用于表示有界时序约束（例如一次定时[≤ 3]）。我们还使用 SALT 的作用域算子来限定需求必须成立的时间，使用之前、之后或之间的事件。作用域区间必须指定是否包含（包含）或排除（排除）其界定事件，以及事件的发生模式：必需（必须发生）、弱（可以发生）或可选（仅当事件发生时才进行评估）。

*4.1.3. FRETish*作用域区间

FRET 将作用域视为端点左侧和右侧之间的区间，具体取决于作用域选项（参见表 2 中名为$left(scope)$ 和$right(scope)$ 的函数）。以下缩写用于描述端点：FTP：执行中的第一个时间点；FiM/LiM：模式期间的第一个/最后一个状态；FNiM/LNiM：不处于模式期间的第一个/最后一个状态；FFiM/FLiM：执行中首次出现 FiM/LiM 的位置。例如，在作用域定义为 FiM 和 LiM 两点之间。

我们重复使用这些端点，但注意到在 PCTL* 的语境下，其中一些端点会重合（参见表 2 中名为 $endpoint(acronym)$ 的函数）。PCTL* 路径公式以未来时线性时序逻辑表示，因此会沿轨迹从左向右向前查看。因此，它们只能在区间左端点出现前一个时间点检测到该端点。例如，当 not (mode) 和 next mode 成立时，就能检测到 FiM（以及 FFiM），这与 LNiM 重合。同样，当 mode 和 not (next not mode) 成立时，就能检测到 LiM （以及 FLiM ），这与 FNiM 重合。因此，大多数作用域区间左开右闭。由于我们对无限轨迹进行推理，轨迹的最后一个时间点会被替换为 false（例如，参见 空值 和之后作用域中的 *右(scope)*），这使得这些区间右端开放。另一种例外情况是左端为 FTP。由于 FTP 是一个点

公式在该点进行解释，因此相应的作用域区间包含 FTP，左端也因此闭合。

4.2. 组合形式化算法

算法1描述了 PCTL* 形式化结果的生成。我们的算法首先从表 2 中检索与时序字段相关联的$tFormula$ （第 2 行）。例如，对于需求 **[P-006]**, , 其中使用下一步 时序，我们检索到 SALT 公式：next incursionDetected，这表示 incursionDetected 必须在下一个时间点成立。接下来（第 3 行），我们强制$tFormula$ 仅在由作用域右侧界定的区间内成立。例如，对于在 作用域中，$ltlform$ 返回：(nextRES)之前（含端点）弱 右侧 ，这确保 next res 必须在作用域区间的右端点之前或恰好在该端点发生（如果该端点出现）。弱运算符确保即使界定事件 LiM 从未发生，该约束仍然有效。注意，$ltlform$ 仅涉及作用域的右侧；检查需求是否在作用域的左侧内得到执行，则由$sform$ 单独处理（见表 2）。

接下来，我们调用 SALT，将$tFormula$ 与作用域选项一并作为输入，并接收一个 LTL 公式作为输出。对于 **[P-006]**, , SALT 返回 `(X incursionDetected) & (!RIGHT)`，这意味着在下一个时间点必须满足检测到侵入，除非当前时间点是作用域区间内的最后一个时间点（右边界）。

在第 4–7 行中，通过考虑概率字段来构造$pFormula$ 。当概率字段为 null 时，$pFormula$ 等于 PCTL* P 运算符内的$ltlFormula$，其概率大于或等于 1。当概率字段不为 null，因而指定了概率界限时，通过将概率P 与预定义的界限运算符应用于$ltlFormula$来表示$pFormula$ 。对于需求 **[P-006]**, , 第 7 行返回 $P_{\geq 0.99}[(\mathrm{XRES})\ \&\ (!\mathrm{RIGHT})]$。

接下来，通过考虑条件字段来构造$baseForm$ （第 8–17 行）。如果条件字段为 null，则将$baseForm$ 设为 probform，这是一个占位符，稍后会被实例化为 $pFormula$ （第 24 行）。根据条件字段设为$holding$ 还是$regular$，条件有两种解释方式。如果条件值为$holding$，$baseForm$ 会在检测到$trigger$ 的任何时刻强制满足该公式，其中$trigger$ 对应于条件（cond）为真。对于 **[P-006]**, , 第 12 行返回 `always(incursionimpliesPROBFORM)`。如果条件值为$regular$，则$baseForm$ 定义为一个合取：第一个合取项规定，只要检测到$trigger$ ，公式就必须成立

表 2：算法1中使用的函数，改编自 [37]。

时序	*tform*（时序）
立即	响应
$next$	下一步 响应
始终	始终 响应
最终	最终 响应
直到	(响应 直到 排他弱停止)
之前	响应释放（不停止）
对于 n	（始终定时⌊≤n⌋ 响应）
在.....以内 n	（最终定时⌊≤n⌋ 响应）
从不	始终(非 响应)
在.....之后	对于(N，非响应) 且在(N+1，响应)以内

作用域	*ltlform*(作用域，*tFormula*)
空值，在.....之后	*tFormula*
在、之前、不在	(*tFormula*) 之前 包含性弱右侧
仅在	不 (*tFormula*) 之前 包含性弱右侧
仅在之前	非（*tFormula*）
仅在之后	非（*tFormula* 之前 包含性弱右侧）

作用域	*sform*(作用域，基本形式)
$null$	基本形式
in	（始终 (基本形式 在排他性可选左侧之间, 包含性弱右侧)) and (模式蕴含 (基本形式 之前 包含性弱右侧))
之前	模式或(基本形式 之前 包含性弱右侧)
之后	(基本形式 之后排他性可选左侧)
不在	在(基本形式)
仅在	在(不 (基本形式))
仅在之前	((非 模式) 蕴含(之后(非(基本形式)))) 并且（模式蕴含（非（基本形式）））
仅在之后	（非（基本形式）之前 包含性弱右侧）

作用域	左(作用域)	右(作用域)
$null$	FTP	假
之前	FTP	FFiM
之后	FLiM	假
in	FiM	LiM
不在，仅在	FNiM	LNiM
仅在之前	FFiM	假
仅在之后	FTP	FLiM

首字母缩略词	端点(首字母缩略词)
FiM,FFiM，LNiM	（非模式）且（下一步模式）
LiM，FNiM，FLiM	（模式）且（下一步非模式）

算法1 概率公式生成

1: **函数** GenProbFormula(作用域, 条件, 概率, 时序)
▷ 时间形式化
2: *tFormula*← *tform*(时序)
3: *ltlFormula*← SALT(*ltlform*(作用域, *tFormula*))
▷ 概率形式化
4: **如果** 概率 =空值 **则**
5: 概率公式 ← $P_{\geq 1}$[*ltlFormula*]
6: **否则如果** 概率 = 界限 **则**
7: 概率公式 ← $P_{\sim \text{BOUND}}$[*ltlFormula*]
▷ 条件形式化
8: **如果** 条件 =空值 **则**
9: 基本形式 ←概率形式
10: **否则，如果** 条件 =保持 **则**
11: 触发条件 ←条件
12: 基本形式 ← 始终(触发条件 蕴含概率形式)
13: **否则，如果** 条件 =常规 **则**
14: 触发条件 ← （（非 条件）且 下一步 条件）
15: *f1* ← **(**始终 **(**触发条件 蕴含 下一步 概率形式**))**
16: *f2* ← **(**条件 蕴含 概率形式**)**
17: 基本形式 ←*f1* 与 *f2*
▷作用域形式化
18: 公式 ← *sform*(作用域, 基本形式)
19: 公式 ← 实例化(公式, 左, 左(作用域))
20: 公式 ← 实例化(公式，右，右(作用域))
21: **对于** 所有 ∈ {FiM、FFiM、LNiM、LiM、FNiM、FLiM} **执行**
22: 公式 ← 实例化(公式，*ac*，端点(*ac*))
23: 公式 ←SALT(公式)
24: 公式 ← 实例化(公式，概率形式，概率公式)
25: **返回** $P_{\geq 1}$[公式]

被检测到（当条件变为真时，即如(`not`条件) `and` `next` 条件)所述，从假转变为真），第二个合取项涵盖触发条件 在区间开始时发生的情况。

接下来，*sform* 函数使用 SALT 作用域算子，强制基本形式 在与指定范围字段相关的所有区间内成立（第 18 行）。请注意，当左侧为 FTP 时，无需在*sform* 中限制左侧的应用范围（例如，作用域之前）。对于作用域*in*，基本形式

表示为合取。第一个合取项指出，在执行轨迹的任意时刻（始终），基本形式 必须在左侧和右侧之间成立，从而在执行过程中所有相关区间上施加基本形式 。第二个合取项适用于模式在轨迹起点成立的情况，即第一个相关区间从 FTP 开始；它指出在这种情况下，基本形式 必须一直成立，直到右侧。之所以需要这个特殊情况，是因为从 FTP 开始的区间左侧并非开区间，如前所述。对于 **[P-006]**, , 第 18 行返回：(始终 (始终(入侵蕴含概率形式) 在.....之间，左侧排除、可选 左侧，右侧包含、弱 右侧))且 (自动_起飞 蕴含(非 (始终 (入侵 蕴含 概率形式))之前，左侧包含、弱 右侧))。

接下来，根据表 2 的规定，将端点 left 和 right（第 19–22 行）替换为给定作用域对应的表达式。对于 **[P-006]**, left（FiM），将其替换为 (非自动_起飞且(下一步 自动_起飞))；对于 right（LiM），则替换为（自动_起飞 且(下一步 非自动_起飞))。在第 23 行，调用 SALT 生成线性时序逻辑表达式。请注意，占位符 probform 直到第 24 行才会被概率概率公式 替换。该算法将所得公式封装在 PCTL* P 运算符中，并在第 25 行返回（见表 4 中的 **[P-006]** ）。

4.3. 算法正确性：终止性与良构性

定理 1. 过程 *Create_Probabilistic_Formula*（算法*1*）对于输入字段作用域、条件、概率和时序的任意有效组合均会终止，并返回一个良构的 *PCTL** 公式（语法定义见第 *4.1.1* 节）。

证明。 我们通过直接证明算法1中的各个步骤，来证明 Create_Probabilistic_Formula 的正确性。

时序（第 ***2–3*** 行）：我们首先证明 $ltlFormula$ 是一个有效的 LTL 公式。为此，只需证明提供给 SALT 函数（第 3 行）的输入是一个有效的 SALT 公式。该函数会调用 SALT 工具 [33] ，生成语法正确的 LTL 公式。我们通过组合 $tform$和 $ltlform$，系统地构造输入 SALT 公式。由于这两个函数（表 2）都是有限的，我们可以穷举所有可能的组合。这样便能通过构造验证，每种组合的输出都是语法有效的 SALT 公式。第 2 行中，$tform$ 的所有可能取值都是有效的 SALT 公式。在表 2 中（以及生成的 SALT 公式中），right、res 和 stop 被

视为原子命题，之后再进行实例化。接下来，我们考察表 2 中 $ltlform$ 的输出（第 3 行）。第一行输出的是有效的 SALT 公式 $tFormula_0$。由于它是第 2 行的输出（如上所述），因此有效。后续各行则通过其他 SALT 构造扩展该公式，例如第二行中的`before inclusive weak RIGHT` 。这些构造各自都会生成有效的 SALT 公式。

概率（第 ***4–7*** 行）：根据概率输入，第 5 行或第 7 行会生成语法有效的 PCTL* 公式（probform），其形式为 $P_{\sim p}[\varphi]$，其中 φ 是一个良构线性时序逻辑公式。[2] 这一构造的正确性基于以下观察：PCTL* 中的概率算子 P 是针对路径公式 Ψ定义的，而每个 LTL 公式 φ在语法上都是路径公式——也就是说，LTL 语法是路径公式片段 Ψ的严格语法子集。

条件（第 ***8–17*** 行）：根据条件输入，基本形式会被赋值为第 9、12 或 17 行中的一个有效 SALT 公式。我们将 COND 和 PROBFORM 视为原子命题，稍后再对其进行实例化。

作用域（第 ***18–25*** 行）：函数 $sform$（第 18 行）首先返回 SALT 公式的一个片段。然后，实例化会将关键字 Left（第 19 行）、Right（第 20 行）以及公式中代表 SALT 片段的任何缩写（第 21–22 行）替换掉。由此可通过构造得到正确的 SALT 公式，其证明方式与第 2–3 行的时序类似。为简洁起见，关于表 2 中首字母缩略词的详细解释，请参阅经典FRET的组合算法 [37] 。在第 23 行，调用 SALT 后，公式会被替换为一个 LTL 公式。请注意，该 LTL 公式包含被视为原子命题的 probform（来自第 9、12 或 15 行）。因此，第 24 行的公式符合以下语法：$\varphi' ::= true \mid a \mid P_{\sim \text{BOUND}}[\text{formulaltl}] \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi_1 \mid X\varphi_1 \mid \varphi_1 U \varphi_2 \mid \varphi_1 U^{\leq n} \varphi_2$。为确保生成的任何公式都符合有效 PCTL*状态公式的语法，第 25 行将其封装在概率算子P中，从而得到一个符合 PCTL* 严格语法子集的公式：

$$\Phi_{formula} ::= P_{\geq 1}[\varphi']$$

终止性：所有操作都是确定性的，并依赖于从表中进行的有限推导、条件分支和变量替换。不存在递归，且第 21 行中的唯一循环恰好迭代 6 次，以处理

[2]我们提醒读者注意 LTL 的语法 [41]：$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi_1 \mid X\varphi_1 \mid \varphi_1 U \varphi_2 \mid \varphi_1 U^{\leq n} \varphi_2$，其中 φ_1和 φ_2都是 LTL 公式。

这组首字母缩略词。因此，每条控制路径都会到达一条 return 语句；假设 SALT 翻译函数终止（参见 [33] 中的 SALT 语义），算法1即可保证终止。

通过分析公式构造步骤，并结合对表 2 和 SALT 翻译函数所生成中间公式的构造正确性论证，我们证明了算法1的最终输出必定是格式良好的 PCTL* 公式。

□

5. 形式化语义的确认

如前所述，我们采用 PCTL*，因为它为概率需求的规范提供了形式语法和语义，例如“系统正确检测到违规的概率（从给定的初始系统状态出发）至少为 96%”（$P_{\geq 0.96}[F\ violation_detected]$）。PRISM [35] 和 Storm [7] 等概率验证工具支持对此类性质进行自动化验证。

为了增强对生成的 PCTL* 公式能够捕捉预期语义的信心，我们采用了专门的验证框架。对于每个模板键及其对应的 PCTL* 公式，该框架都会检查公式是否符合该模板键的语义。作为确认过程的一部分，我们生成离散时间马尔可夫链（DTMC），将其用作针对生成公式的测试模型。

我们的确认方法受到成熟软件工程技术的启发，例如基于预言机的测试和覆盖率驱动测试。为确保全面覆盖，我们针对全部 560 个 FRET 模板键生成测试用例，并为每个模板键测试多种字段取值。我们的方法借鉴了 Prospec [20] 和经典FRET [37]（二者均针对线性时序逻辑）所采用的确认方法，这些方法依赖轨迹生成以及 Spin/NuSMV 模型检测。不过，我们的方法引入了一项关键扩展：构造 DTMC 模型，并使用 PRISM 进行概率模型检测，从而将成熟的确认技术拓展到概率领域。尽管并非穷举，我们的方法仍能系统地探索模板键空间，并允许我们重用经典FRET 预言机，以确保语义与经典FRET 基于线性时序逻辑的定义保持一致。

我们首先简要介绍离散时间马尔可夫链，然后详细说明我们的确认方法。

5.1. 预备知识：离散时间马尔可夫链（*DTMC*）

离散时间马尔可夫链描述了一个在离散时间步 [42]之间，于有限个可数状态构成的集合中转移的系统。离散时间马尔可夫链形式化定义为一个元组 $\langle S, s, P, L\rangle$，其中 S是有限状态集， $s \in S$ 是初始状态， $\overline{P}$：$S \times S \to [0,1]$ 是转移概率矩阵，L：$S \to \overline{2}^{AP}$ 是一个将状态标记为 AP中原子命题的函数。注意，对于所有 $s \in S$，$\Sigma_{s' \in S} P(s, s') = 1$。如果离散时间马尔可夫链当前处于状态 s，那么 $P(s, s')$ 定义了在一个时间步以内转移到 s' 的概率。离散时间马尔可夫链的一条路径是由有限或无限个状态组成的序列，描述了所建模系统的一种可能执行。

5.2. 自动化验证框架

我们介绍该框架的三个主要组件。概率模型生成器（图 1）组件采用随机方法生成 DTMC 模型；图3展示了一个示例。模型的每个状态$s0$–$s9$均由状态变量元组（t, m, c, sc, r）定义，分别表示时间、模式、条件、停止和响应。时间 $t \in [0,$最大值$]$,最大值 $\in \mathbb{N}$ 表示到达状态 s∈{s0, s1, . . . , s9} 所需的步数。例如，到达状态 s6 需要 5 步，因此 $t = 5$ 在 s6 中。其余状态变量均为布尔值 m， c， sc， $r \in [0, 1]$。例如，在状态 s9 中，cond 和 stop 为 0，而 mode 和 res 为 1。

为了生成离散时间马尔可夫链，我们的工具首先为Max随机选择一个数字。以我们的示例$Max = 6$为例。其次，对于每种模式、条件、停止和响应，它会随机选取 0 到 3 个（随机、互不相交且不连续的）区间，范围从 0 到Max。在我们的示例中，为模式构造了两个区间，即区间 [0,1] （状态 $s0 - s1$）和区间 [3,6] （状态 $s3 - s9$）；为条件构造了一个区间，即 [3,5]）（状态 $s3 - s7$）；为停止构造了零个区间。响应区间的构造采用不同的方法：我们像之前一样选择随机区间，但由于响应可能具有概率性，因此将 r 分别设为 0 和 1，并置于不同的分支状态中。在示例中，选择了区间 [4,6]，并据此生成了概率转移，例如从 s3 转移到 $s4 - s5$、从 s4 转移到 $s6 - s7$，以此类推。最后，我们的工具会生成一个随机转移概率矩阵。

概率预言机组件用于解释离散时间马尔可夫链上模板键的语义，并生成一个预期值，该值可以为真或假。我们的离散时间马尔可夫链不包含循环（最终状态中的自环除外），因此每个模型都表示有限条路径。

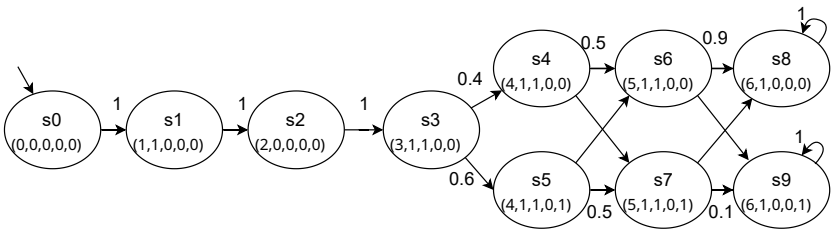

图3：用于确认的生成离散时间马尔可夫链示例。状态由一个五元组 (t, m, c, sc, r) 定义，分别表示时间、模式、条件、停止和响应。第一个元素 t 通过计算从初始状态开始的转移次数来跟踪时间。其余四个元素是模式 (m)、条件 (c)、停止 (sc) 和响应 (r) 的布尔指示量，其中 1 表示该命题在该状态下为真，0 表示为假。

我们的方法如下。对于每个离散时间马尔可夫链：

1. 枚举所有可能的路径。2. 对不含概率字段的模板键应用现有的[3] 经典 FRET 预言机 [37]。如果预言机返回 true，则保留该路径；否则将其丢弃。3. 对所得子集，计算每条保留路径的子路径概率。子路径的概率是该子路径各转移概率的乘积。

概率预言机如算法2所示，以组合方式通过上述步骤3中保留的路径计算子路径的概率。它计算的子路径表示一系列状态：从模式和条件（如果有）成立的状态开始，直到响应结束。每条子路径的概率都必须满足需求的概率约束。

对于通过离散时间马尔可夫链的保留路径，设 s 为其状态序列s_i，其中 $0 \leq i < length(s)$。设 $last = length(s) - 1$，并设 $prob_i$ 为从状态 s_{i-1} 转移到状态 s_i的概率，其中 $prob_0$ =1。因此，子路径的概率为 $\prod_{i=l+1}^{e} prob_i$ （第6行），其中 l 是子路径的左（起始）索引，e 是其右（结束）索引。

起始索引 l 由条件和模式的类型决定。当不存在条件或模式时，l 为 0。例如，当存在一个

[3]我们还为保持条件创建了一个新的非概率预言机。

算法2 概率预言机

```
1: 函数 CheckProbability(离散时间马尔可夫链，关键字，概率界)
2:     不良路径 ← ∅；良好路径 ← ∅
3:     对于所有 路径 ∈ dtmc.paths.filter(λp.FREToracle(p，关键字) 执行
4:         对于所有 l ∈ 起始于(路径，关键字.作用域，关键字.条件) 执行
5:             e ← ⟨表3⟩(路径 l，关键字.时序)
6:             e ← getRightIndex(路径 l，关键字。时序)                    请参见表 3
7:             p ← 归约(∗，[l + 1 . . . e].映射(离散时间马尔可夫链。概率))
8:             如果 概率界(p) 则
9:                 良好路径 ← 良好路径 ∪ {路径}
10:            else
11:                不良路径 ← 不良路径 ∪ {路径}
12:    返回 [良好路径，不良路径]
```

保持条件 c以及模式期间 m时，可能存在多个子路径，每个子路径都从索引 i 开始，其中 $s_i \models c \wedge m$。对于正规条件c 以及模式期间 m的情况，可能存在多个子路径，每个子路径都从索引i 开始，其中 $s_i \models c \wedge m$，并且 $i = 0$ 或 $s_{i-1} \models \neg c$。不同模式和条件组合对应的子路径也以类似方式定义。清单中的 starts 函数（第 4 行）计算给定路径的左索引集合。

结束索引 e由起始索引 l 和时序约束类型决定。我们用 r 表示响应，用 t 表示在.....以内和for 时序中的时间界，并用 sc 表示之前和直到时序中的停止条件。函数 getRightIndex（第 6 行）以路径、起始索引 l和来自关键字.时序的时序约束为输入。它利用该时序约束在表 3 中查找对应的行，然后应用该行中的公式来计算右索引 e。例如，如果时序为“下一步 r”，函数会查找表中的第二行，并计算 $e = \min(last,\ l + 1)$。对于“最终 r”这类更复杂的时序，公式会将 e 确定为满足 r 的第一个索引，并确保子路径中没有更早的状态满足 r。

对于直到约束，第一个析取项处理停止条件 sc 从不成立的情况；因此，响应 r 必须从索引 l开始一直到路径 $last$末尾都成立。第二个析取项将 e 定义为响应 r 必须成立的最后时刻，也就是停止条件 sc 成立之前的时刻。

该概率语义评估器 使用 PRISM 模型检查器。它接收离散时间马尔可夫链模型、PCTL* 公式以及预期值作为输入，该预期值由

表 3：右子路径索引的计算。函数 getRightIndex(路径 l，时序) 根据时序类型查找适当的公式并计算右索引，e。第一列显示 FRET需求的不同时间选项。第二列提供用于计算从左索引 l 开始的子路径的右索引 e 的公式，该子路径位于路径中。此处， $last$ 表示路径的最终索引， s_i 表示路径中索引为 i 的状态， r 表示响应命题， sc 表示停止条件， t 表示时间界。

时序	右索引（记作 e）
立即 r	$e = l$
下一步 r	$e = \min(last,\ l+1)$
始终 r	$e = last$
从不 r	$e = last$
最终 r	$l \le e \wedge (s_e \models r)\ \wedge \forall k\,.\, l \le k < e \Rightarrow s_k \models \neg r$
直到 $sc\ r$	$l \le e \wedge (((\forall k\,.\, l \le k \le last \Rightarrow s_k \models \neg sc)\ \wedge e = last)$ $\vee((s_{e+1} \models sc)\ \wedge \forall k\,.\, l \le k \le e \Rightarrow s_k \models \neg sc))$
之前 $sc\ r$	$l \le e \wedge (s_e \models r)\ \wedge \forall k\,.\, l \le k < e \Rightarrow s_k \models \neg r$
对于 $t\ r$	$e = \min(last, l+t)$
在.....以内 $t\ r$	$l \le e \le l+t \wedge (s_e \models r)\ \wedge \forall k\,.\, l \le k < e \Rightarrow s_k \models \neg r$
之后 $t\ r$	$e = \min(last, l+t+1)$

概率预言机提供。它使用 PRISM 语言对离散时间马尔可夫链模型进行编码，并评估公式的真值。最后，它检查 PRISM 返回的真值是否与概率预言机生成的预期值一致。

确认高亮。 尽管我们在形式逻辑方面经验丰富，但我们的确认框架对于发现算法生成公式中的错误至关重要。这表明，编写旨在供分析工具直接使用的公式时，不应仅仅依赖人工完成。

我们发现的一个细微错误涉及 $ltlform$ 函数返回的 SALT 公式初始版本，其中我们错误地使用了可选修饰符，而非预期的弱修饰符。例如，$ltlform$ 最初返回的是 $(tFormula)$ 之前（含）可选 右，用于在 作用域。

乍一看，这种表述似乎合理，因为“可选”和“弱”都暗示作用域区间的右端点可能无法到达。然而，两者的语义存在细微却至关重要的差异。

可选修饰符引入了条件约束：如果到达指定端点，就必须满足所述行为；如果未到达，则不施加任何约束。相比之下，弱修饰符要求无论最终是否到达该端点，都必须满足这一性质。对于允许区间永不结束的作用域，可选修饰符施加的约束不足，导致对预期行为的下近似。

由于公式在结构上相似，这种差异在人工检查时很容易被忽略。我们的验证框架帮助我们发现并纠正了这一问题。

6. 评估

本文提出的扩展是由公开可用的工业案例研究中的需求所推动的，包括消防场景中的可靠性需求 [43]和自主系统需求 [44, 45]。接下来，我们通过研究两个问题（灵感来自 [46]），评估 FRETish 在描述实际场景中的需求时的有效性。我们还将在第三个研究问题中评估 FRETish 模板键的实用性。

RQ1：文献中的概率需求在多大程度上可以用扩展 FRETish 表达?

研究问题2：扩展版 FRET 能否用于描述复杂工业场景中的需求?

RQ3：针对**RQ1** 和**研究问题2**收集的需求中，最常用的是哪些模板键?

请注意，我们并不关注工具的执行时间，因为由于采用基于缓存检索的机制，翻译本身速度很快，通常能在几毫秒内完成。

实验设置。 为回答上述每个研究问题，我们整理了以下四组需求：

集合1：该需求集是通过对来自多个交通运输行业和国防领域的工业真实场景需求进行详细调查而获得的 [28]。我们直接从 [28] 的作者处获取了该集合（共133项需求）。

集合2：我们对 Google Scholar[4]，中引用了 [28]，的239篇同行评审论文进行了广泛的文献综述，提取出199项自然语言需求和形式化概率需求。

集合3：我们考察了定量验证基准集（QVBS）中公开提供的 38 个 PCTL*性质 [47] 。该基准集提供了一个丰富的资源库，涵盖形式化指定需求、相关概率模型，以及多种工具的验证结果。

Set4：我们与行业
合作伙伴 RTRC 合作，整理出一组包含 21 项需求的需求。这些需求用于针对其概率模型进行验证，该模型描述了其自主飞机滑行、起飞与着陆系统。

这些集合既包含文献、公共代码库和工业用例中的自然语言需求，也包含形式化需求。它们涵盖多个领域，包括生物过程和医疗任务、软件应用以及信息物理系统。我们重点关注可用 PCTL* 表达的需求，即本研究的目标逻辑。因此，我们排除了依赖其他逻辑构造的性质，例如 CSL 的稳态算子或 RPCTL 的奖励。

6.1. 在实际场景中表达概率需求（*RQ1*）

RQ1旨在考察我们的 FRETish 扩展在多大程度上能够涵盖现有的公开概率需求。为此，我们研究了集合1、集合2和集合3。

集合1：在来自 [28]，的 133 条 PCTL* 需求中，我们的 FRET 概率扩展能够表达 108 条。其余 25 条因以下原因而不受支持：

- 22 使用有界直到算子，或使用除 [0,N], 以外的时间区间，而经典 FRET（因此我们的概率扩展也）不支持这些情况。我们的扩展版 FRET 继承了这些经典 FRET 的限制。

- 两个涉及嵌套概率，这些目前尚未由该扩展处理。

[4]https://scholar.google.com/

- 一个使用概率结构（$P_{\geq 1}$ 位于 $P_{\sim \text{界限}}$内），本文称之为反向概率，这也超出了所支持的语法范围。

集合2：我们表达了引用 [28]的论文中199项需求里的175项。24项不受支持的需求包括：有界直到（12）、嵌套概率（2）、除 [0,N] 以外的时间区间（5）、嵌套时序算子（2）和反向概率（3）。

集合3：我们表达了38项 QVBS 需求中的30项。8项不受支持的需求涉及有界直到（7）或嵌套时序算子（1），这些在经典 FRET 中均不受支持。

为回答**RQ1**，概率 FRETish 使我们能够表达现有公开可用需求中的绝大多数（334/391，$\sim 85\%$）。值得注意的是，大多数不受支持的情况（57项中有44项）涉及有界或嵌套时序算子——这些限制继承自经典 FRETish。其中包括若干使用了不同于 [0...N] 的有界时间区间的需求。例如，考虑集合1中的以下需求：“排队网络在*0.5*到*2*个时间单位内变满的概率低于*0.1*”。该需求可用 PCTL*形式化为 $P < 0.1[F[0.5, 2]\,\text{满}],$，但经典 FRET 不支持诸如 $[0.5, 2]$ 这样的时态界限，我们的概率扩展也继承了这一限制。未来工作计划解决这些不足。

6.2. 工业场景中的概率需求（研究问题*2*）

为回答**研究问题2**，我们在 RTRC 提供的航空领域工业案例研究中评估了所提出的扩展。该案例研究聚焦于一种感知系统，该系统根据传统传感器和学习使能组件（LEC）传感器的输入生成单一传感器输出，这些传感器协同用于各种飞机机动，例如自主滑行、起飞和着陆。

图4展示了 RTRC 提供的简化感知系统架构。为了估计环境参数 k 的值，系统会处理各种传感器输入（例如摄像机图像），并输出一组参数值 $\hat{q}_k$。这些输入会经过一系列（M+N）估计流水线 来处理原始输入数据（例如用于 LEC 传感器的卷积神经网络（CNN）组件）。每个流水线都会输出一个估计值$\hat{q}_{k,\ i|j}$ 并集成故障检测逻辑 [48] ，用于输出布尔值

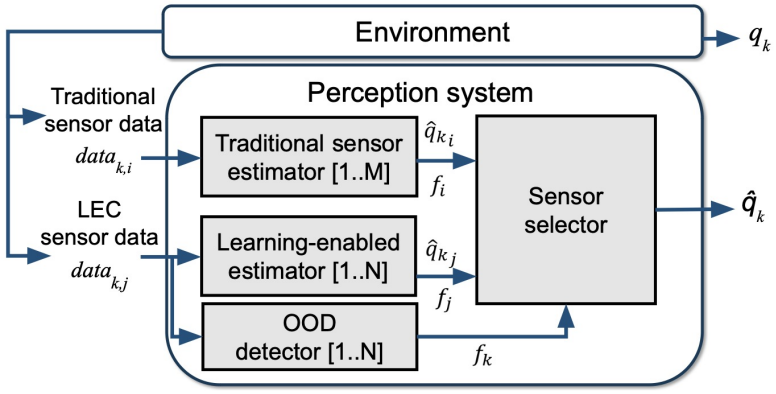

图4：由 RTRC 提供的简化感知系统概览。该系统会在时间 k 生成一个对 q 的估计值 $\hat{q}$。多个传感器（传统传感器和 LEC）采集数据，再由一系列估计器进行处理。LEC 组件可通过分布外（OOD）检测器增强；当预测到分布外值时，检测器会设置标志 f_k。传感器选择器会整合这些估计值，生成一个单一估计值 $\hat{q}_k$。

变量 $f_{i|j}$ 以指示传感器故障。LEC 组件还包含一个分布外（OOD）检测器，当传感器数据属于分布外数据时，会生成布尔标志 f_k。随后，传感器选择组件会组合这些估计值，得出最终估计值 $\hat{q}_k$，供控制器或应急管理系统使用。通过将估计值 $\hat{q}_k$ 与实际真实值 q 进行比较，评估预测准确度。

在本案例研究中，我们成功将工业合作伙伴 RTRC 提供的全部 21 条自然语言需求（Set4）表示出来。表 4 展示了其中一部分。虽然这些需求大多是概率需求，但其中有 2 条不是（例如，表 4 中的 **[P-001]** ）。对于这些需求，我们的 FRET 扩展生成了 LTL 和 PCTL*；后者被 RTRC 用于针对其 PRISM 模型进行分析。

我们的结果表明，FRET 能够有效支持在这一工业案例研究中对概率需求进行规范。通过与合作伙伴多轮讨论（数月内累计约 30 小时），我们验证了形式化结果与自然语言需求的预期含义相符；合作伙伴在形式逻辑方面具有专业知识。工业合作伙伴的反馈总体上是积极的。

> “概率 *FRET* 让普通工程师无需具备太多形式化知识，也能轻松将需求形式化为 *PCTL**

表 4：我们的工业案例研究中的 FRETish 和生成的 PCTL* 公式。完整列表见 [13]。

ID	自然语言	FRETish	模板 Key	PCTL*
[P-001]	传感器选择 组件 *shall* 始终 满足准确的 理想条件。	每当 理想条件 传感器选择 shall 立即 满足q_hat = q	[空值, 保持, 空值, immediately]	P>=1 [(G (理想条件 => (P>=1 [(q_帽 = q)])))]
[P-006]	在自动起飞模式下, 每当跑道 发生跑道侵入 , *the* 传感器选择输出 *shall* 以概率 > *0.99* 在下一时间点 满足跑道侵入检测 跑道侵入。	在自动_起飞_模式下 每当 q_k 传感器选择 shall 概率为 > 0.99 在下一个时间点 满足检测到侵入	[in, 保持, 界限 , 下一步 |	P>=1 [((G ((! (((! 自动_起飞_模式) & (X 自动_起飞_模式)))) | (X ((自动_起飞_模式 & (X (! 自动_起飞_模式))) R (q_k => (P>0.99 [((自动_起飞_模式 & (X (! 自动_起飞_模式))) | ((X 检测到侵入) & (! (自动_起飞_模式 & (X (! auto_起飞 _模式))))))])))))) & (自动_起飞_模式 => ((自动_起飞_模式 & (X (! 自动_起飞_模式))) | ((自动_起飞_模式 & (X (! 自动_起飞_模式))) R (q_k => (P>0.99 [((自动_起飞_模式 & (X (! 自动_起飞_模式))) | ((X 检测到侵入) & (! (自动_起飞_模式 & (X (! auto_起飞 _模式))))))]))))))]
[P-007]	自动着陆模式之后, 传感器选择 输出 *shall*, 且 概率 > *0.99* 最终满足 正确检测 跑道出口	在自动_着陆_模式之后 传感器选择 shall 概率为 > 0.99 最终 满足 检测_正确_退出	[之后, 空值, 界限 , 最终]	P>=1 [(((! (自动_和_模式 & (X (! 自动_和_模式)))) U ((自动_和_模式 & (X (! 自动_着陆_模式))) & (X (P>0.99 [(F 检测_正确_退出)])))) | (G (! (自动_着陆_模式 & (X (! auto_着陆_模式))))))]
[P-012]	发生跑道 侵入时, 跑道 侵入探测器 , 以大于 *0.99%* 的概率, 检测到 侵入之前达到 不安全间隔 距离。	当 q_k 跑道侵入检测器 shall以概率 > 0.9999 之前 不安全_sep_距离 满足 检测到侵入	[空值, 常规, 界限 , 之前]	P>=1 [((G (((! q_k) & (X q_k)) => (X (P>0.9999 [(检测到侵入 R (! 不安全_sep_距离))])))) & (q_k => (P>0.9999 [(检测到侵入 R (! 不安全_sep_距离))]))]
[P-017]	每当发生跑道 侵入, RunwayIn- 入侵检测器 , with 概率大于 （99.99%）, 检测到 在 10 个时间单位之前发生的入侵 个时间单位。	每当 q_k 跑道侵入检测器 shall的概率为 > 0.9999在 10 个时钟周期以内 满足检测到侵入	[空值, 保持, 界限 , 在.....以内]	P>=1 [(G (q_k => (P>0.9999 [(F<=10 检测到侵入)])))]
[P-019]	一旦准确, 跑道探测器 其概率大于 99% 时仍能保持 准确 10 个时间 单位。	准确时 跑道检测器shall 概率为 > 0.99 持续 10 个时钟周期满足 q_hat =q	[空值, 常规 界限, for]	P>=1 [((G (((! 准确) & (X 准确)) => (X （P>0.99 [(G<=10(q_hat = q))])))) & （准确的 => （P>0.99 [(G[<=10] (q_hat = q))])))]

形式化方法背景。这也将大幅加快具备形式化方法背景的工程师形式化需求的进程。”

他们还概述了未来改进的若干方向。

> “和任何语言一样，*FRET* 也需要一段学习时间，用户才能熟练地编写出始终符合预期的 *PCTL**。不过，学习时间并不长。语言中的某些部分，例如条件和模式，需要深入理解其语义，这超出了新手用户的直观认知。模拟器和相关说明对于理解语言中的一些细微之处很有帮助。”

探索这些易用性方面将成为我们未来 FRET 项目的核心内容。具体而言，我们计划将经典FRET的图示说明扩展到概率需求，使确认过程对不同背景的用户而言更加直观、易于使用。

6.3. 扩展 *FRETish* 的实际效用（*RQ3*）

为回答 RQ3，我们分析了集合1–4中概率 FRETish需求的结构。图 5展示了各需求集中所使用的模板键的分布和多样性。图 6重点展示了最常用的模板键。如表 5所示，四个需求集共计334项需求中，有69.8%仅遵循图 5和图 6所示的五种不同模式。有趣的是，这些模式均未使用范围字段，尽管334项需求中有14项（4.2%）使用了作用域。其中一半来自 RTRC 案例研究（14项需求中的7项），占集合4的33%。尽管先前文献 [28] 很少包含作用域（全局作用域除外），但其他使用经典 FRETish 的大规模研究表明，作用域的使用更为频繁 [49, 50], ，我们的工业示例也是如此。我们与行业合作伙伴讨论了文献中作用域并不常见这一现象，对方指出“这很令人惊讶，因为模式在控制和自主系统软件中很常见”。 这与我们的经验一致，即模式经常出现在关键系统中，因此 FRET 表达特定模式需求的能力在实践中尤其有价值。

表 5：常用模板键。

模板键	集合1	集合2	集合3	Set4	总计
[空值，空值，界限，在……以内]	40/108	31/175	13/30	0/21	84/334 (25.1%)
[空值，空值，界限，最终]	19/108	40/175	11/30	0/21	70/334（21%）
[空值，常规，界限，在……以内]	18/108	20/175	0/30	1/21	39/334（11.7%）
[空值，空值，界限，直到]	8/108	12/175	5/30	0/21	25/334 (7.5%)
[空值，保持，界限，在……以内]	5/108	8/175	0/30	2/21	15/334（4.5%）

我们的贡献还包括为 FRETish新增了一种条件值（保持条件，每当构造）。这种构造出现在334项需求中的34项（10.2%）中，凸显了其实用价值。值得注意的是，图 5和图 6展示了需求在模板键之间的分布，其中相当一部分（主要来自集合4）归入“其他”类别。这表明，尽管某些模板键的使用频率高于其他模板键，但这些需求集使用了种类繁多的模板键，总计达到43个模板键。

7. 讨论

7.1. 遇到的挑战

为了扩展 FRET 以支持概率需求，我们必须仔细权衡设计方案，并应对若干非平凡的挑战。

纳入概率语义。 我们必须考虑如何支持概率。添加新的概率字段看似自然，但也引发了一些问题：该字段应位于需求结构中的什么位置，才能支持细致的概率规范，同时避免空泛满足？确定准确的位置需要详细分析其与现有字段之间的交互（见第3节第3.1小节）。

从线性时序推理转向分支时序推理。 我们不得不重新思考 FRET 的推理方法。经典FRET依赖线性时序逻辑（LTL），沿单条执行轨迹解释需求满足。然而，概率推理自然适合概率计算树逻辑（PCTL）等分支时序逻辑，这类逻辑考虑多个可能未来的分布。与此同时，还必须保留 FRETish 的原始设计，这使挑战更加复杂，并且因

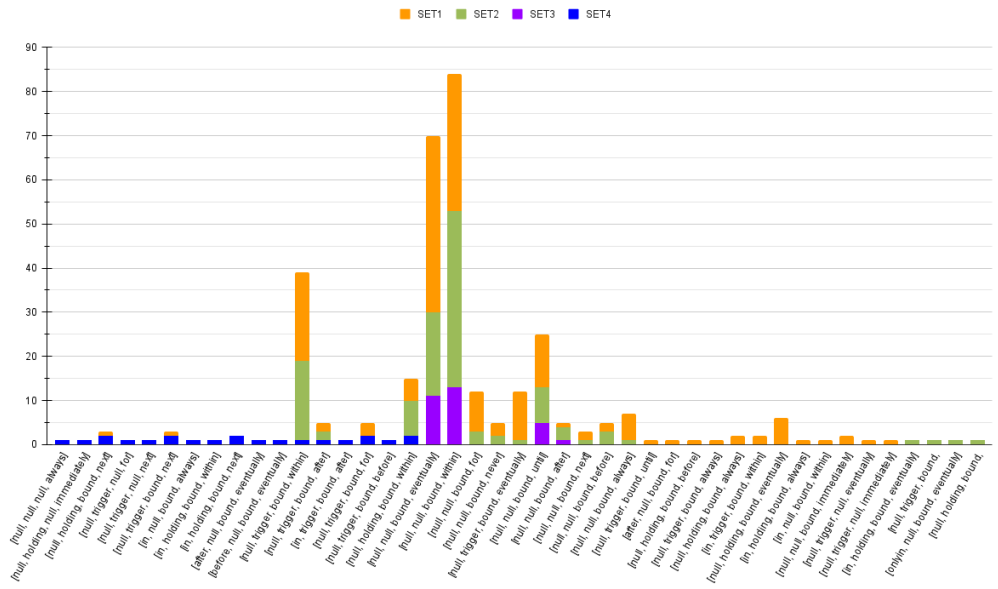

图5：每个需求集中各模板键的出现总次数。

与预先存在的 FRETish 字段，尤其是范围字段的集成而进一步加剧。经过多次迭代，我们通过引入两步分解（见算法1）解决了这一挑战。在开发该分解的过程中，我们发现了容易引入错误的细微语义差异。为确保正确性，我们开发了专门的验证框架，以确认所得形式化公式忠实地表达了预期的概率语义（见第5节）。

支持研究文献中观察到的概率模式。我们确定，文献中描述的许多常见概率模式无法仅使用现有的 FRET 构造和新的概率字段来表达。在不破坏 FRET 语言设计的前提下纳入这些模式颇具挑战，需要探索多种替代方案，包括扩展 FRET 字段和修改模板键。最终，我们引入了一种新的条件类型，事实证明，该类型对于表达广泛的概率模式至关重要，同时还能保持与现有语言构造的一致性（见第3节）。

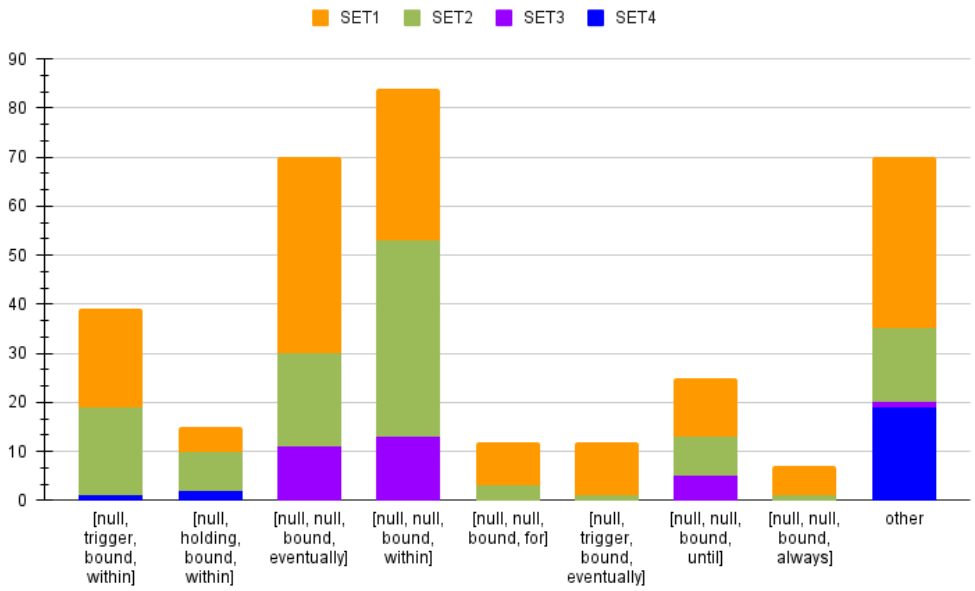

图6：按需求集汇总的每个模板键出现次数总览。此处，我们将出现次数为 ≤ 6 的模板键移至“其他”类别。

模板空间的规模。FRETish 模板键的数量为语言的表达能力提供了一种定量度量，其中每个键都代表一种明确且形式化定义的模式。因此，模板键数量的大幅增加意味着 FRET 捕获更广泛需求的能力得到了实质性扩展。我们的扩展将 FRET 支持的模板键总数从160个增加到560个。这一扩展后的集合大幅增加了扩展 FRET 所需的工作量。具体而言，为了最终确定形式化算法，我们手动推导并审查了 $150+$ 个公式。此外，模板空间的规模使得开发自动化验证框架成为必要，而在验证过程中发现的形式化问题则要求我们分析220多项测试结果。

与现有$FRET$ 工具链的集成。最后，确保 FRET 的概率扩展能够与现有 FRET 功能无缝集成并保持向后兼容性至关重要。这要求所有核心功能（例如模板选择、需求验

证、形式化翻译以及形式化结果的生成）都能与概率规范正确配合。

7.2. 结构化自然语言作为大语言模型（*LLM*）与形式化规格之间的中介

大语言模型（LLM）凭借先进的自然语言处理能力，正开始应用于需求工程，立即带来了效率提升。然而，LLM 本身具有不确定性，且容易产生幻觉，这与该领域对精确性的需求形成了鲜明矛盾。因此，FRET 等工具仍不可或缺，尤其是在安全关键领域，在这些领域中，人们不鼓励依赖 LLM，或对其依赖实行严格监管。即使在不那么关键的场景中，使用结构化中间表示（如 FRETish）也能提升 LLM 的效果。

在我们的一个项目中，我们发现，先将含糊的自然语言需求转换为 FRETish，再将其转化为形式化规格，相较于直接一步转换为线性时序逻辑等时序逻辑，能显著提高输出准确率。我们的发现与 [51]，中的研究结果一致：与直接翻译方法相比，采用结构化中间表示可以将翻译正确性提高 18–43%。此外，使用 FRETish 这样的结构化语言不仅能提高翻译准确率，还能让从业者利用 FRET 内置的可解释性和形式化分析能力 [10]。在 FRETish 层面进行操作，工程师能够获得有关需求一致性的可解释反馈，及早发现歧义，并系统地管理复杂的规格约束。综上所述，这些观察结果凸显了我们为概率需求正式定义的结构化自然语言的相关性与及时性。

8. 效度威胁

内部威胁。 尽管我们的确认方法并非穷尽所有情况，但我们通过生成 7000 多个模型（其中 FRETish 字段的赋值各不相同），并为算法 4.2 提供正确性的形式化证明，来降低这一威胁。

外部威胁。我们整理了四个不同的需求集来评估研究问题，每个需求集都经过选择，旨在提供互补的视角和覆盖范围。尽管经过了精心构造，某些局限性仍可能

影响研究结果的普遍适用性。集合1源自一篇具有开创意义的 2008年出版物 [28]。尽管该出版物至今仍有重要影响，但可能未能涵盖较新的进展。集合2在此基础上纳入了所有引用 [28], 的论文中的需求，反映了活跃研究社群中思想的发展演变。更广泛的文献综述可以进一步丰富该需求集。集合3取自一个成熟的基准，这是软件工程中的标准做法，不过可能会遗漏罕见或新兴的模式。集合4是在与工业合作伙伴合作的过程中开发的，确保了其实用价值。未来的工作可以考虑与来自不同领域的更多合作伙伴合作，进一步扩大覆盖范围。

我们总共收集了334条概率需求。虽然我们已努力确保该语料库兼具多样性和代表性，但它可能尚不足以支持广泛的泛化。尽管如此，这仍是我们工作的一项具体贡献，并为未来研究和评估中不断扩充的概率需求基准奠定了基础。

9. 结论

随着关键系统日益复杂，并纳入不确定的传感器数据和不可预测的机器学习组件，软件需求必须不断演进以捕捉不确定性，这一点已愈发显而易见。在必须遵循严格的形式化开发、验证和确认流程的领域，这种演进尤为必要。

为此，我们提出了一种新颖的自动化框架，用于将结构化自然语言中的概率需求形式化，并将其作为 FRET 的扩展实现。我们的贡献包括一种组合算法，该算法由专门的验证框架和正确性证明提供支持；此外，我们还利用来自不同领域、基于文献和工业需求的语料进行了全面评估。值得注意的是，我们的工具能够表达大多数需求；少数例外源于经典FRET的局限，或涉及某些形式的概率嵌套。这些发现验证了我们方法的实用性，并为支持 RPCTL 等其他逻辑指明了方向。

致谢

A. Mavridou 由美国国家航空航天局合同编号 80ARC020D001 资助。M. Farrell 获得英国皇家工程院研究奖学金的资助。G. Vázquez 在 KBR Inc. 实习期间完成了本研究的部分工作。

在 NASA 艾姆斯研究中心。T. Pressburger 获得空域运行与安全计划下的 NASA 系统级安全项目资助。NASA 大学领导力倡议（拨款编号 # 80NSSC20M0163）提供经费，支持 T.E. Wang 开展这项研究。R. Calinescu 获得英国高级研究与发明局的安全 AI 计划以及约克自主性保障中心的资助。M. Fisher 获得 EPSRC 拨款 EP/X02489X/1 资助的 CRADLE（面向严苛且持久运行环境的机器人自主性中心）以及英国皇家工程院新兴技术讲席计划的资助。

参考文献

[1] M. Fisher, R. C. Cardoso, E. C. Collins, C. Dadswell, L. A. Dennis, C. Dixon, M. Farrell, A. Ferrando, X. Huang 等，《巡检机器人验证与确认挑战概述》，《机器人学》10 (2) (2021) 67。[2] L. Rierson，《开发安全关键软件：航空软件与DO-178C合规性的实用指南》，CRC Press，2017年。[3] M. Farrell, A. Mavridou，《探索学习型软件的需求：研究预览》，载于：需求工程：软件质量基础国际工作会议，施普林格，2023年，第179–188页。[4] M. Farrell, M. Luckcuck, O. Sheridan, R. Monahan，《为需求而烦恼：飞机发动机控制器的形式化需求》，载于：需求工程：软件质量基础国际工作会议，施普林格，2022年，第96–111页。[5] K. Y. Rozier，《规范：形式化方法与自主性面临的最大瓶颈》，载于：经验证软件：理论、工具与实验工作会议，施普林格，2016年，第8–26页。[6]M. Kwiatkowska, G. Norman, D. Parker，《PRISM 4.0：概率实时系统的验证》，载于：G. Gopalakrishnan, S. Qadeer（编），第23届国际计算机辅助验证会议（CAV’11）论文集，LNCS第6806卷，施普林格，2011年，第585–591页。

[7] C. Dehnert, S. Junges, J.-P. Katoen, M. Volk, 《风暴将至：现代概率模型检验器》，载于：国际计算机辅助验证会议，施普林格，2017年，第592–600页。[8] E. M. Hahn, Y. Li, S. Schewe, A. Turrini, L. Zhang, 《iscasMC：基于网络的概率模型检验器》，载于：国际形式化方法研讨会，施普林格，2014年，第312–317页。[9]L. Grunske, P. Zhang, 《监测概率属性》，载于：第7届欧洲软件工程会议与ACM SIGSOFT软件工程基础研讨会联合会议论文集，ESEC/FSE '09，计算机协会，纽约，美国，2009年，第183–192页。doi：10.1145/1595696.1595724。[10] D. Giannakopoulou, A. Mavridou, J. Rhein, T. Pressburger, J. Schumann, N. Shi, 《使用FRET进行形式化需求获取》，载于：需求工程：软件质量基础，2020年，第1页。[11] A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, A. L. Sangiovanni-Vincentelli, 《通常都能奏效：随机系统的时序逻辑》，载于：计算机辅助验证：第7届国际会议，CAV'95，列日，比利时，1995年7月3日至5日，论文集7，施普林格，1995年，第155–165页。[12] C. Baier, 《概率系统的算法验证方法》，获得曼海姆大学数学与信息学学院 venia legendi 资格的特许讲师资格论文（1998年）。[13]A. Mavridou, M. Farrell, F. G. Vazquez, 《概率FRET GitHub网站与案例研究》，https://github.com/NASA-SW-VnV/fret/releases/tag/v3.0.0, https://github.com/Gricel-lee/FRET-Anonymous（2025年）。[14] N. Feng, L. Marsso, S. G. Yaman, B. Townsend, A. Cavalcanti, R. Calinescu, M. Chechik, 《迈向规范性需求获取的形式化框架》，载于：2023年第38届IEEE/ACM国际自动化软件工程会议（ASE），IEEE，2023年，第1776–1780页。[15]A. Sorathiya, G. Ginde, 《从应用评论中提取与伦理关切相关的软件需求》，载于：第39届

IEEE/ACM国际自动化软件工程会议，2024年，第2251–2255页。[16] M. Fazelnia、M. Mirakhorli、H. Bagheri，翻译巨擘，推理挑战：用于检测冲突需求的可满足性辅助语言模型，载于：第39届IEEE/ACM国际自动化软件工程会议论文集，2024年，第2294–2298页。[17] K. Kolthoff、C. Bartelt、S. P. Ponzetto、K. Schneider，借助自动化图形用户界面原型设计进行需求自 elicitation，载于：第39届IEEE/ACM国际自动化软件工程会议论文集，2024年，第2354–2357页。[18] M. B. Dwyer、G. S. Avrunin、J. C. Corbett，有限状态验证中性质规范的模式，载于：第21届国际软件工程会议论文集，1999年，第411–420页。[19] S. Konrad、B. H. Cheng，实时规约模式，载于：第27届国际软件工程会议论文集，2005年，第372–381页。[20] I. Gallegos、O. Ochoa、A. Gates、S. Roach、S. Salamah、C. Vela，用于生成形式化规范的性质规约工具：Prospec 2.0，载于：第20届国际软件工程与知识工程会议（SEKE 2008），2008年，第273–278页。[21] S. Konrad、B. H. C. Cheng，促进基于规约模式的性质的构造，载于：第13届IEEE国际需求工程会议（RE 2005），2005年8月29日至9月2日，法国巴黎，2005年，第329–338页。doi:10.1109/RE.2005.29。[22] A. W. Fifarek、L. G. Wagner、J. A. Hoffman、B. D. Rodes、M. A. Aiello、J. A. Davis，SpeAR v2.0：需求的过去时线性时序逻辑规约形式化与分析，载于：NASA形式化方法——第9届国际研讨会，NFM 2017，美国加利福尼亚州莫菲特机场，2017年5月16日至18日，论文集，2017年，第420–426页。doi:10.1007/978-3-319-57288-8_30。[23] A. Mavin，倾听，然后使用EARS，《IEEE软件》29（2）（2012年）17–18。doi:10.1109/MS.2012.36。

[24] B. Jeannet、F. Gaucher，使用STIMULUS调试嵌入式系统需求：汽车案例研究，载于：第8届欧洲嵌入式实时软件与系统大会（ERTS 2016），法国图卢兹，2016年，第1页。[25] K. Siu、A. Moitra、M. Durling、A. Crapo、M. Li、H. Yu、H. Herencia-Zapana、M. Castillo-Effen、S. Sen、C. McMillan等，使用ASSERTTM开发飞行关键软件与系统，载于：2017年IEEE/AIAA第36届数字航空电子系统会议（DASC），IEEE，2017年，第1–10页。[26] A. Crapo、A. Moitra、C. McMillan、D. Russell，ASSERT(TM)中的需求捕获与分析，载于：2017年IEEE第25届国际需求工程会议（RE），2017年，第283–291页。doi: 10.1109/RE.2017.54。[27] S. Getir Yaman、P. Ribeiro、A. Cavalcanti、R. Calinescu、C. Paterson、B. Townsend，自主代理的社会、法律、伦理、共情与文化需求的规范、确认与验证，《自主代理的共情与文化需求》（2023年）。[28] L. Grunske，概率质量属性的规格模式，载于：第30届国际软件工程会议论文集，2008年，第31–40页。doi:10.1145/1368088.1368094。[29] M. Autili、L. Grunske、M. Lumpe、P. Pelliccione、A. Tang，使用结构化英语语法统一定性、实时与概率性质规范模式，《IEEE软件工程汇刊》41（7）（2015年）620–638。[30] V. Forejt、M. Kwiatkowska、G. Norman、D. Parker，概率系统的自动验证技术，载于《永恒网络化软件系统的形式化方法：第11届计算机、通信与软件系统设计形式化方法国际学校，SFM 2011》，意大利贝尔蒂诺罗，2011年6月13日至18日。高级讲座11（2011年）53–113。[31] C. Menghi、C. Tsigkanos、M. Askarpour、P. Pelliccione、G. Vázquez、R. Calinescu、S. García，移动机器人的任务规范模式：支持定量属性，《IEEE软件工程汇刊》

《软件工程》49 (4) (2023年) 2741–2760。doi:10.1109/TSE.2022.3230059.[32] G. V. Flores, A. Mavridou, M. Farrell, T. Pressburger, R. Calinescu，《机器人学》：FRET需求的新任务，载于：NASA形式化方法，施普林格，2024年，第359–376页。[33] A. Bauer, M. Leucker，SALT的理论与实践，载于：M. Bobaru, K. Havelund, G. J. Holzmann, R. Joshi（编），NASA形式化方法，施普林格柏林海德堡，柏林，海德堡，2011年，第13–40页。[34] Z. Manna, A. Pnueli，《反应式与并发系统的时序逻辑：规范》，第1卷，施普林格科学与商业媒体，1992年。[35] M. Kwiatkowska, G. Norman, D. Parker，PRISM性质规范简介，https://www.prismmodelchecker.org/manual/PropertySpecification/Introduction，访问日期：2022-03-22（2022年）。[36] M. Farrell, A. Ferrando, M. Xu，BDI智能体的定量运行监测，载于：国际自主智能体与多智能体系统会议，计算机协会，2025年，第2517–2519页。[37] D. Giannakopoulou, T. Pressburger, A. Mavridou, J. Schumann，结构化自然语言需求的自动形式化，《信息与软件技术》137 (2021) 106590。doi:https: //doi.org/10.1016/j.infsof.2021.106590。[38] T. Parr，《ANTLR 4权威参考》，Pragmatic Bookshelf，2013年。[39] D. Giannakopoulou, T. Pressburger, A. Mavridou, J. Schumann，基于结构化自然语言生成形式化需求，载于：需求工程：软件质量基础：第26届国际工作会议 REFSQ 2020，意大利比萨，2020年3月24–27日，会议论文集26，施普林格，2020年，第19–35页。[40] E. A. Emerson, J. Y. Halpern，重新审视“有时”与“并非从不”：论分支时序逻辑与线性时序逻辑，《ACM学报》（JACM）33 (1) (1986) 151–178。

[41] A. Pnueli, 《程序时序逻辑》，载于：第18届计算机科学基础年度研讨会（SFCS 1977），IEEE，1977年，第46–57页。[42] M. Kwiatkowska, G. Norman, D. Parker，随机模型检测，《性能评估形式化方法：计算机、通信与软件系统设计形式化方法第7届国际学校》（2007年）220–270。[43] T. Pressburger, A. Mavridou, I. Perez, A. Goodloe, P. Mehlitz, D. Vol- pano, H. Walsh，野火扑救用例需求监测，技术报告，美国国家航空航天局（2023年）。[44] N. Benz, I. Sljivo, P. G. Vlastos, A. Woodard, C. Carter, M. Hejase, 《Troupe系统：自主多智能体火星车集群》，载于：AIAA SCITECH 2024论坛，2024年，第2894页。[45] E. Asaadi, S. Beland, A. Chen, E. Denney, D. Margineantu, M. Moser, G. Pai, J. Paunicka, D. Stuart, H. Yu，航空平台上基于机器学习的自主性的可靠集成，载于：2020年AIAA/IEEE第39届数字航空电子系统会议（DASC），IEEE，2020年，第1–10页。[46] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, T. Berger，机器人任务的规范模式，《IEEE软件工程汇刊》47 (10) (2019) 2208–2224。[47] A. Hartmanns, M. Klauck, D. Parker, T. Quatmann, E. Ruijters，定量验证基准集，载于：系统构造与分析工具和算法国际会议，施普林格，2019年，第344–350页。[48] M. Bonfe, P. Castaldi, W. Geri, S. Simani, 通用航空飞机机载传感器的故障检测与隔离，《自适应控制与信号处理国际期刊》20 (8) (2006) 381–408。[49] M. Farrell, M. Luckcuck, R. Monahan, C. Reynolds, O. Sheridan，FRET规范与形式化建模：机械肺呼吸机，载于：严格状态建模方法国际会议，施普林格，2024年，第360–383页。

[50] A. Mavridou、H. Bourbouh、D. Giannakopoulou、T. Pressburger、M. He- jase、P.-L. Garoche、J. Schumann，《洛克希德·马丁十大网络物理挑战：形式化、分析与阐释》，载于：第28届IEEE国际需求工程会议论文集，2020年，第300–310页。

[51] M. Macedo、Y. Tian、P. Nie、F. R. Cogo、B. Adams，《Intertrans：利用传递式中间翻译提升基于大语言模型的代码翻译》，载于：2025年第47届IEEE/ACM国际软件工程会议（ICSE），IEEE，2025年，第1153–1164页。