arXiv:2603.20356v1 [cs.LO] 20 Mar 2026

Agentproof：智能体工作流图的静态验证

Melwin Xavier Vaisakh M A Lule°理工大学，瑞典 独立研究者
melwin.xavier@ltu.se 997vaisakh@gmail.com
MelveenaJolly Midhun Xavier 独立研究者 独立研究者
melveenajollyk@gmail.com midhun@industriagents.com

摘要

智能体框架越来越多地将工具使用行为编码为显式工作流图，但安全执行仍是一个运行时问题。这些框架通过其 API 提供可分析的图结构，使得能够在部署前静态验证安全属性，而运行时护栏只能对此进行被动检查。

本文介绍了 AGENTPROOF，这是一个系统，可从四种主要智能体框架（LangGraph、CrewAI、AutoGen、Google ADK）中自动提取统一抽象图模型，执行六项结构检查并生成见证轨迹，还可通过编译为确定性有限自动机的领域特定语言评估时序安全策略——既通过图与确定有限自动机的积构造进行静态评估，也在运行时基于事件轨迹进行评估。与通用模型检查器不同，Agentproof 无需手动建模。

在一个包含 18 个由作者构建的工作流的精选基准中，27% 的工作流存在结构缺陷（死端节点、不可达出口），而在执行人工闸门策略时，55% 的工作流违反了该策略——这是两类不同的问题，先前的研究却将它们混为一谈。定义的 15 项时序策略均可用七种形式的领域特定语言片段表示；对于最多包含 5,000 个节点的图，验证可在不到一秒的时间内完成。该语料库旨在作为评估静态验证工具的可复现基准，而非一项流行率研究；缺陷率反映的是工具在定向基准上的检测能力，而非生产系统中的基准发生率。尽管如此，静态图验证仍可作为运行时护栏的补充，捕捉运行时工具无法发现的拓扑层面缺陷，除非实际执行了存在问题的路径。

1 引言

大语言模型（LLM）正日益被部署为能够与外部工具和 API 交互的自主智能体 [27, 33]。这些智能体不再只是孤立地生成文本，还会检索文档、查询数据库、发送消息，并触发会产生现实世界副作用的下游工作流操作。越来越多的研究对这一范式进行了综述 [31], ，同时也涌现出若干编排框架，将智能体行为组织为明确的工作流图：LangGraph [4],CrewAI [2], AutoGen [32], 和 Google ADK [3]均将计算步骤表示为节点，将允许的转换表示为边。

尽管这种结构化表示十分丰富，当前智能体系统中的安全执行仍主要是运行时 层面的问题。NeMo Guardrails [23] 和 LlamaGuard [18]等工具会拦截单次大语言模型调用或工具调用，并在执行时应用内容级或策略级过滤器。虽然这些工具有助于检测有害输出和提示注入尝试，但它们

运行时方法有三个根本局限：（i）它们会带来每次调用的延迟开销；（ii）只有在实际执行到违规路径时，才能检测出违规行为；（iii）对于测试期间未触发的路径，它们无法提供任何覆盖保证。

示例。 考虑一个电子邮件分流智能体，其工作流如下：根据意图对收到的电子邮件进行分类，将其路由至紧急或普通处理程序，起草回复并发送。在迭代开发过程中，开发者在普通优先级路径上添加了一个回复草稿节点，却忘记将其连接到发送节点。结果出现了一个*死端节点*：普通优先级电子邮件进入起草阶段后便悄然停止。除非使用会触发该缺陷的电子邮件来执行普通优先级路径，否则运行时护栏无法发现这一缺陷。然而，对工作流图进行静态分析可以立即标记出这个死端节点，并生成一条见证轨迹：开始 →分类 → 路由器 → 普通处理程序 → 起草回复（卡住）。

研究空白。 内容级运行时护栏无法解决这类缺陷——工作流图中的拓扑错误。原则上，经典模型检验技术能够检测此类错误：可达性分析、死锁检测和时序逻辑验证在形式化方法文献中早已得到广泛应用 [11, 7]。然而，SPIN [17] 和 NuSMV [10] 等通用模型检查器要求手动将待分析系统转换为专用建模语言，这对于不断迭代工作流设计的智能体开发者而言，开销大到难以承受。类似地，业务流程管理（BPM）社区长期以来一直致力于工作流健全性验证 [29], ，但这些方法针对的是可视化建模工具和标准化表示法（例如 BPMN、佩特里网），而非智能体框架用来定义其图的程序化 API 对象。迄今为止，尚无系统能够直接从框架源代码中自动提取并静态验证智能体工作流图。

拟议方法。 本文主张对智能体工作流图进行*部署前静态验证*。关键在于，现代智能体框架通过其 API 暴露工作流结构，使图可以自动提取和分析。当智能体行为以图结构表示时，安全属性便可归结为对拓扑属性进行可达性、隔离性和时序约束检查，而无需运行智能体即可穷尽验证 [11]。

本文介绍了 Agentproof，这是一个用于智能体工作流静态分析的实用系统，重点关注以下性质：(a) 可在工作流图上表达；(b) 独立于大语言模型文本生成语义。与通用模型检查器不同，Agentproof 会从框架源代码中自动提取分析模型，无需手动建模。

贡献。

- 跨框架提取流水线：针对四种主要智能体框架（LangGraph、CrewAI、AutoGen、Google ADK）的提取器，可自动弥合异构表示之间的差异——这些表示各有不同的入口/出口约定、边语义和层级模型——并将其转换为统一的、包含带类型节点和边的抽象工作流模型，从而免去通用模型检查器所需的手动建模步骤（第5节）。
- 一个部署前验证流水线，包含六项结构检查以及见证轨迹生成，并采用时序策略领域专用语言，覆盖 LTL 的安全片段，编译为确定性有限自动机（第6节）。

- 对18个精选工作流进行的实证评估表明，结构缺陷（死节点、不可达出口）和策略违规（缺少人工关卡）十分常见；对于最多包含5,000个节点的图，验证耗时不到一秒（第7节）。

范围。 拟议方法会及早验证工作流设计决策，例如：“此图能否到达破坏性工具？”或“在两个敏感操作之间是否必须进行人工审查？”该方法并不试图证明大语言模型输出的语义属性（例如真实性），因为这些属性本质上仍具有非确定性。威胁模型（第 3节）对这一界限进行了明确讨论。

2 背景

2.1 Agentworkflow 图

许多智能体框架会将行为表示为有向图，有时也称为状态图或工作流图。节点对应计算步骤：大语言模型调用、工具调用、根据状态进行分支的路由器／选择器，或人工审批关卡。边对应步骤之间的状态转移，包括条件分支、并行扇出和循环回边。

这种显式的控制流模型是图编排智能体与单体式“单提示”应用的关键区别：无需执行智能体，即可对图结构进行静态分析。

2.2 结构验证

结构属性仅取决于图拓扑和类型注释。典型示例包括：(i) 可达性（能否从入口点到达某个节点？）、(ii) 死胡同（没有出向转换的节点）、(iii) 隔离性（敏感工具之前是否必须经过关卡节点？），以及 (iv) 路由器节点的合理性约束。

这些检查开销很低：可以使用标准图算法（广度优先搜索／深度优先搜索）实现，并在与 $|V| + |E|$ 成线性关系的时间内运行。

2.3 时序监控器

线性时序逻辑 (LTL) 等时序逻辑 [25] 提供了一种形式化语言，用于描述执行轨迹上的约束。一种常见的验证策略是将时序约束编译为自动机，并对执行轨迹或系统模型与自动机的积 [30]进行推理。

Agentproof 将时序策略领域专用语言编译为确定性有限自动机，并支持两种互补的评估模式：(i) 通过图 $\times$ 确定有限自动机积构造进行静态验证，无需执行即可检查图中的所有路径；(ii) 通过运行时监控，在实时或模拟的事件流上评估确定有限自动机。该领域特定语言面向 LTL 的安全片段，特意以牺牲完整表达能力为代价，换取部署简便性。

3 威胁模型

智能体工作流安全面临三类威胁：

T1：开发者错误。开发者可能会无意中创建包含结构缺陷的工作流：不可达出口节点、会无声地中断执行的死端分支、边类型错误的路由器，或缺少显式声明的工具节点。这些缺陷自然产生于迭代式开发过程中，且难以通过人工检查发现，尤其是在大型图中。

T2：恶意工作流注入。 拥有工作流定义写入权限的对手（例如通过遭到入侵的配置文件、共享的工作流注册表，或针对工作流库的供应链攻击）可能会构造绕过安全门的拓扑结构——例如，添加一个条件分支，使流程绕过必需的人工审批节点。

T2 的信任假设。 部署模型假设采用这样的架构：Agentproof 作为可信 CI/CD 流水线中的强制验证门运行（例如，在 GitHub Actions 工作流中，或作为构建系统的部署前钩子）。对手可以修改纳入版本控制的工作流定义文件，但不能 修改 CI/CD 流水线配置、验证工具或提取器实现。这类似于静态代码检查器如何检测开源项目中的恶意代码贡献：代码检查器是可信的，贡献则不可信。

T2 未涵盖的内容。 若对手入侵构建系统本身、Agentproof 二进制文件或框架运行时，就能完全绕过验证。此类攻击需要在基础设施层面提供完整性保护（代码签名、安全启动、访问控制），不在本工作的范围之内。

实践中的注意事项。 目前尚未在实际环境中观察到针对智能体工作流注册表的 T2 攻击。之所以纳入此类威胁，是因为攻击面在理论上确实存在（共享代码仓库、可通过 pip 安装的工作流包），而且防范这类攻击无需增加任何机制，只需进行 T1 所需的结构检查。我们尚未开展对抗性评估，以确定复杂攻击者能否构造出绕过全部六项结构检查、同时仍能实现恶意目标的拓扑；开展此类红队测试是未来研究的重要方向。

T3：运行时图变更。 部分框架允许在执行期间动态修改工作流图（例如，向团队添加智能体、修改转移规则）。如果修改后的图未经重新验证，那么部署时确立的保证可能不再成立。

信任边界。 本方法假设框架 API 语义（例如 LangGraph 的 StateGraph、CrewAI 的 Crew）是正确的：提取器能够忠实读取框架将要执行的图结构。Python 运行时和提取器实现均属于可信计算基。开发者编写或修改的工作流定义 以及运行时生成的大语言模型输出 均不受 信任。

范围。 结构检查和时序监控器以静态方式应对威胁 T1 和 T2：它们会在部署前检测有缺陷或恶意的拓扑结构。威胁 T3 则需要运行时重新验证或不可变工作流定义；相关内容将在第 9 节讨论。大语言模型的输出内容（例如提示注入、有害生成）明确不在范围内；这由互补的运行时护栏处理（第 8 节）。

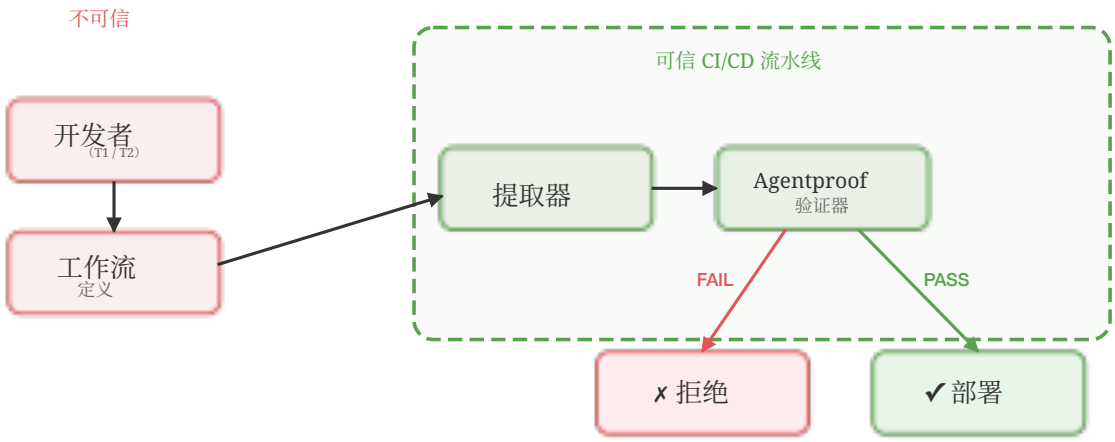


图1：T2 的部署模型：Agentproof 作为 CI/CD 验证门。工作流定义不可信；提取器、验证器和流水线均属于可信计算基。

表1：威胁类别和 Agentproof 覆盖范围。

威胁	示例	静态	运行时
T1：开发者错误	死端节点	✓	
T1：开发者错误	不可达出口	✓	
T1：开发者错误	缺少人工关卡	✓	
T2：注入	绕过审批	✓	
T2：注入	添加不安全的工具路径	✓	
T3：变更	动态添加智能体	重新验证 *	✓
	有害大语言模型输出		✓

*如果在变更后重新验证图，即可解决。

4 系统概述

图 2 总结了 Agentproof 的工作流程。该系统接收特定框架的智能体定义，提取抽象图模型，然后运行一套结构检查和时序检查。

步骤 1：获取工作流对象。该方法面向将控制流以图或组合树形式公开的智能体编排库。Agentproof 为 LangGraph、Google ADK、AutoGen 和 CrewAI 提供提取器 [4, 3, 1, 2]。

步骤 2：提取 AgentGraph。每个框架都有一个提取器，可生成通用表示：由带类型的节点和带类型的边组成的集合，并附带可选元数据（例如，绑定到节点的工具名称）。这使下游分析不依赖于特定框架。

步骤 3：运行结构检查。 结构检查可检测无需执行智能体即可发现的工作流缺陷：不可达节点、死胡同、路由形状不匹配，以及策略要求时是否存在人在回路节点。

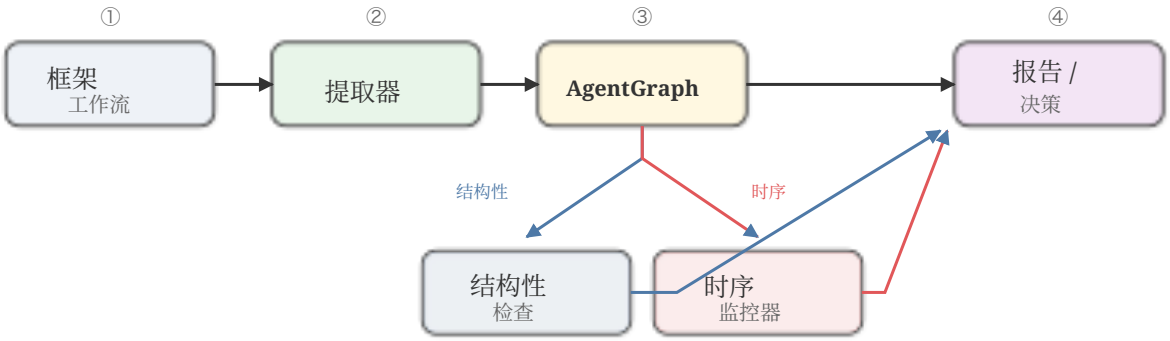


图 2：Agentproof 流水线：将框架工作流提取为抽象图模型，执行结构验证，并将时序策略编译为用于轨迹评估的监控器。

步骤 4：编译时序策略。 时序策略领域专用语言会被编译为确定有限自动机。策略针对抽象事件（工具调用、动作标签或决策）编写，而不是针对框架内部实现。

步骤 5a：静态时序验证。 通过积构造将每个编译后的确定有限自动机与提取图结合：在 $(V \times Q)$ 上执行广度优先搜索，以检测沿<该图的任意>路径可达的时序违规，而无需执行智能体。

步骤 5b：运行时监控器评估。 监控器也会在离线模拟轨迹上进行评估，或集成到运行时事件流中。违规会映射到不同的处理级别（警告、阻止、停止、升级），以支持不同的运行策略。

5 图模型与提取

本节定义了一种框架无关的工作流表示，其中包含带类型的节点和边。这种抽象是所有验证的共同基础。

5.1 形式化模型

定义 1（智能体工作流图）。智能体工作流图是一个元组 $G = (V,\ E,\ \kappa_V,\ \kappa_E,\ T,\ v_0,\ V_f)$ 其中：

- V 是一个有限节点集。
- $E \subseteq V \times V$ 是一个有向边集。
- $\kappa_V : V \to \mathcal{K}_V$ 为每个节点指定一种类型，其中 $\mathcal{K}_V = \{\textsc{entry},\ \textsc{exit},\ \textsc{tool},\ \textsc{llm},\ \textsc{router},\ \textsc{human},\ \textsc{subgraph},\ \textsc{passthrough}\}$。
- $\kappa_E : E \to \mathcal{K}_E$ 为每条边指定一种类型，其中 $\mathcal{K}_E = \{\textsc{direct},\ \textsc{conditional},\ \textsc{parallel},\ \textsc{loop}\}$。

- T： $V \to 2^{ToolNames}$ 将每个节点映射到一个（可能为空的）已声明工具名称集合；仅当 $\kappa_V(v) = \text{TOOL}$时， $T(v)$ 才非空。
- $v_0 \in V$ 其中 $\kappa_V(v_0) = \text{ENTRY}$ 是唯一的入口节点。
- $V_f \subseteq V$ 其中 $\kappa_V(v) = \text{EXIT}$ 对所有 $v \in V_f$ 而言，都是出口节点的集合。

定义 2 (执行轨迹**)**。 G 的轨迹 是一个有限序列 $\pi = v_0 v_1 \dots v_n$，满足$(v_i, v_{i+1}) \in E$ 对所有 $0 \le i < n$均成立。$\text{Traces}(G)$ 表示所有极大轨迹的集合——即终止于 V_f 中的节点，或没有出边的节点的轨迹。进入某个循环且无法到达 V_f或死胡同的轨迹不是极大轨迹；反向可达性检查（ExitReachAll）可检测此类活锁条件。

定义 3 (结构谓词**)**。 定义了 G 上的六个结构谓词：

1. ExitReach(G*)* $\iff \forall v_f \in V_f.\, v_f \in$ *Reach(*v_0*)*，其中 *Reach(*v_0*)* 是从 v_0 经有向路径可达的节点集合。*2. ExitReachAll(*G*)* $\iff$ *Reach(*v_0*)* $\subseteq$ *RevReach(*V_f*)*，其中 *RevReach(*V_f*)* 是可经有向路径到达 V_f 中某个出口节点的节点集合。这保证每个可达节点最终都能到达一个出口。*3. NoDead(*G*)* $\iff \forall v \in V.\, \kappa_V(v) \neq \text{EXIT} \implies \exists u \in V.\, (v, u) \in E$。*4. RouterShape(*$G$*)* $\iff \forall v \in V.\, \kappa_V(v) = \text{ROUTER} \implies \forall (v, u) \in E.\, \kappa_E(v, u) = \text{CONDITIONAL}$。*5. HumanGate(*$G$*)* $\iff \exists v \in V.\, \kappa_V(v) = \text{HUMAN}$。一种可选的基于路径的变体 *HumanGateCov(*G, S*)*，验证对于指定的一组敏感工具 S，从 v_0 到调用 S 中某个工具的节点的每条路径，都经过一个 HUMAN 节点。*6.* 工具声明*(*G*)* $\iff \forall v \in V.\, \kappa_V(v) = \text{TOOL} \implies T(v) \neq \emptyset$。

每个谓词都可在与 $|V| + |E|$成线性关系的时间内判定。附录 A 中给出了可靠性引理，证明通过检查即可保证相应的轨迹级性质。

定义 *4*（抽取正确性）。对于工作流对象 W，若提取图 $G = \mathcal{E}_F(W)$ 满足以下条件，则称提取器 $\mathcal{E}_F$ 对框架 F 是正确的：*(i)* G 中的每个节点都对应于 W 中的一个计算步骤（没有虚构节点）；*(ii)* W 中的每个计算步骤都作为一个节点出现在 G 中（没有遗漏节点）；*(iii)* $(u, v) \in E$ 当且仅当 W 允许从对应于 u 的步骤转移到对应于 v 的步骤。条件 *(i)–(ii)* 对应于节点精确率/召回率 $= 1$；条件 *(iii)* 对应于边精确率/召回率 $= 1$。第 *7.7* 节对此进行了实证验证。

5.2 框架提取器

Agentproof 包含可将框架对象映射到此模型中的提取器：

LangGraph。 LangGraph 暴露了一个包含哨兵起始/结束节点的编译后的状态图。提取器会识别入口/出口哨兵，根据工具绑定和命名启发式（例如人工审核节点）推断节点类型，并在底层框架将某条边标记为条件边时，对其进行编码。

Google ADK。 Google ADK 将智能体组合成由顺序、并行和循环智能体构成的树。提取器执行树遍历，为复合智能体生成子图节点，并对顺序链、并行扇出和循环回边进行编码。

AutoGen。 AutoGen 的团队拓扑可以通过团队对象（例如轮询）或智能体显式列表加转移关系来提供。提取器会构建发言者转移的有向图，合成入口/出口，并将叶发言者连接到出口。

CrewAI。 CrewAI 描述任务管线（顺序式或层次化）。提取器会为每项任务创建一个节点，根据流程模式添加顺序边或管理器路由边，并纳入任务之间的上下文依赖。

5.3 跨框架提取挑战

一项核心贡献是从异构框架 API 中自动提取 AgentGraph 实例。各框架表示工作流的方式各不相同，因此统一提取并非易事：

- **LangGraph** 使用显式的 StateGraph，并调用 add_node() 和 add edge()。入口和出口是字符串哨兵（START、END）。条件路由使用 add conditional edges(), -并配合路径映射字典。
- **CrewAI** 通过传入 Crew 的 Task 对象列表隐式定义工作流。拓扑取决于 process 参数（顺序式或层次化）以及任务之间可选的上下文依赖。
- **AutoGen** 采用群聊模式（RoundRobinGroupChat、SelectorGroupChat），由智能体排序定义图。轮询循环会产生循环边。
- **Google ADK** 通过嵌套构造器（SequentialAgent、ParallelAgent、LoopAgent）和子智能体列表来组合智能体。必须将层次结构展平为单一图，并为其指定适当的边类型。

提取器会将这些不同的表示规范化为相同的 `AgentGraph` 类型系统（定义1），并在框架未明确提供入口/出口时合成入口/出口哨兵节点。SPIN [17]或 NuSMV [10]等通用模型检查器也能验证相同的性质，但开发者必须 手动 将工作流转换为 Promela 或 SMV；即使是小型图，这一过程也要花费数小时（具体示例见第 8 节）。

5.4 工作流图示例

图3 展示了一个小型提取工作流，本文将始终使用它来说明带类型的节点和条件分支。

6 验证方法

Agentproof 提供两个互补的验证层：(i) 对提取图进行结构检查，以及 (ii) 基于事件流进行评估的时序监控器。

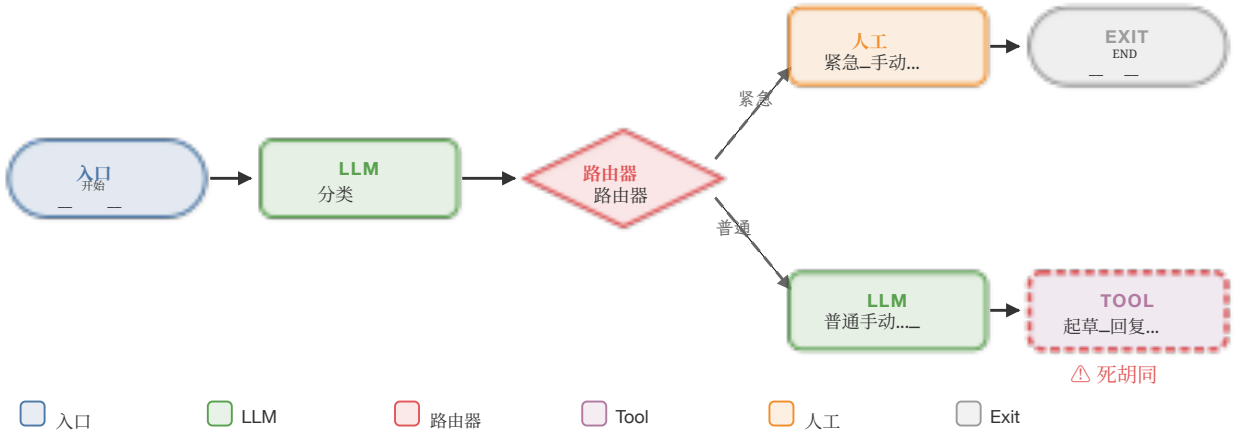


图 3：带类型节点的工作流图提取示例（入口/路由器/大语言模型/工具/人工/出口）。

6.1 结构检查

结构检查基于 G （定义 1）中的邻接结构和节点/边类型标签进行。每项检查都实现了定义 3中的一个谓词。

出口可达性。 使用广度优先搜索（BFS）计算从 v_0 可达的节点集合，以验证$V_f \subseteq \text{Reach}(v_0)$。复杂度： $O(|V| + |E|)$。

反向可达性。 另一项检查用于验证从 v_0 可达的每个节点都能到达 V_f中的某个出口节点。算法从所有出口节点出发，在反向 邻接结构上（沿边的反方向遍历）执行广度优先搜索，以计算 $\text{RevReach}(V_f)$。$\text{Reach}(v_0) \setminus \text{RevReach}(V_f)$ 中的任何节点都是活锁节点：它们从入口可达，却被困在无法通向终止的循环中。复杂度： $O(|V| + |E|)$。

死端检测。 节点 v 满足 $\kappa_V(v) \neq \text{EXIT}$ 且没有出边。死胡同表明可能缺少状态转移或错误处理不完整。使用预计算邻接结构时，复杂度为 $O(|V| + |E|)$。

路由器形状检查。该检查会验证从 ROUTER 节点发出的所有出边都带有标签CONDITIONAL。复杂度： $O(|E|)$ 。

人在回路中的存在性。 存在性检查用于确定图中是否包含任何具有 $\kappa_V(v) = \text{HUMAN}$ 的节点。可选的 覆盖率 变体用于验证：对于一组 S 敏感工具名称，不存在一条从 v_0 到调用 S 中某个工具的节点的路径能够避开所有 HUMAN 节点。覆盖率检查会构建一个排除 HUMAN 节点的修改后的邻接结构，并从 v_0 执行广度优先搜索；任何敏感

工具节点仍然可达，都表明存在一条无人工路径。复杂度：存在性 $O(|V|)$；覆盖率$O(|V| + |E|)$。

工具声明　　检查。 该检查会标记具有 $T(v) = \emptyset$ 的 TOOL 个节点。复杂度： $O(|V|)$。

见证轨迹生成。 检查失败时，验证器会生成一条见证轨迹：一条具体的图路径，用于展示缺陷。对于不可达出口，见证轨迹会展示从 v_0 到可达性边界以及不可达目标的路径。对于死胡同，见证轨迹是从 v_0 到受阻节点的广度优先搜索路径。见证轨迹遵循标准模型检测实践 [11]，能显著帮助调试。

静态时序验证。 除了运行时轨迹评估外，还可以通过图与确定有限自动机的积构造对时序策略进行静态检查。对于确定有限自动机状态为 Q、转移函数为 δ 的已编译规则，乘积状态空间为 $V \times Q$。从 (v_0, q_0) 执行广度优先搜索，在每个节点根据其事件签名推进确定有限自动机，并沿图的边继续搜索。如果任何乘积状态 (v, q') 在确定有限自动机违规状态集中具有q' ，则该属性被违反，广度优先搜索父链会给出一条见证路径。已探索状态的数量上限为 $|V| \cdot |Q|$，因此对于固定规模的确定有限自动机，验证的复杂度仍与图规模呈线性关系。

6.2 时序策略监视器

Agentproof 支持一种时序策略领域特定语言，涵盖线性时序逻辑的安全片段 [25], ，可编译为确定性有限自动机，并以每个事件恒定的开销进行评估。

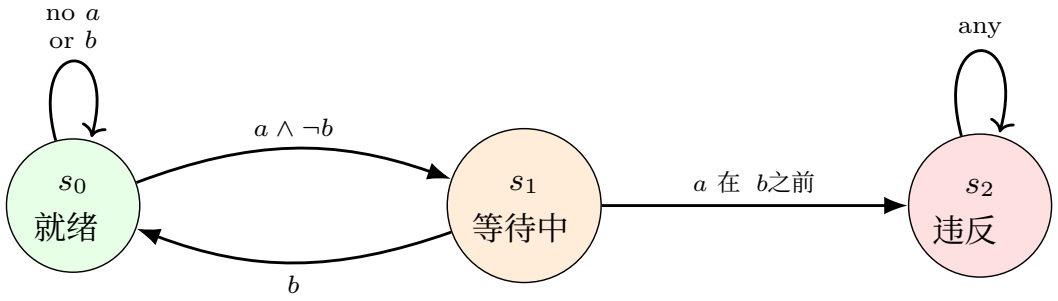
事件模型。 事件是包含可选字段的字典：工具名称、动作类型、决策以及一组标签。谓词会针对这些字段进行匹配（例如，tool:delete account）。事件符号 $\sigma(v)$ 根据节点类型和工具绑定生成：工具节点会针对每个$t \in T(v)$发出 tool:t ； 其他节点则发出其类型标签。

领域专用语言语法。 该领域专用语言支持七种表达式形式（完整的巴科斯-诺尔范式见附录B）：

1. **禁止：** G !atom：只要 atom 曾经出现，就会违反。2. **未来蕴含模式：** a -> F b：如果 a 出现，那么在 a 再次出现之前，b 必须出现。3. **直到模式：** a U b：在 b 出现之前，每一步都必须满足 a。4. **有界响应：** a -> F [<=k] b：如果 a 出现，那么 b 必须在 k 步内出现。5. **响应链：** a -> F b -> F c：一旦 a 出现，在 a 再次出现之前，b 必须先出现，随后 c 必须按顺序出现。6. **合取：** (expr) AND (expr)：两个子性质必须同时成立。7. **析取：** (expr) OR (expr)：至少一个子性质必须成立。

表2：按模式类型划分的确定有限自动机状态数量。

模式	状态
G !原子命题	2
a -> F b	3
a U b	3
a -> F[<=k] b	$k+2$
a -> F b -> F c（ n的链）	$n+1$
（A）与（B）	$\lvert Q_A\rvert \times \lvert Q_B\rvert$
（A）或（B）	$\lvert Q_A\rvert \times \lvert Q_B\rvert$



示例领域特定语言： $a \rightarrow \mathbf{F}\, b$ (交错约束)

图4：用于表示交错约束的确定有限自动机示例：在 a 发生后， b 必须在 a 重复之前发生。

编译为确定有限自动机。 每个表达式都会被解析为抽象语法树，并通过直接构造法编译为确定有限自动机。基础模式（禁止、未来蕴含模式、直到模式）会生成具有 2–3 个状态的确定有限自动机。有界响应会生成计数器增强型确定有限自动机，包含 $k+2$ 个状态。合取和析取使用标准的积自动机构造，生成包含 $|Q_L| \times |Q_R|$ 个状态 [30]。确定有限自动机转移以预计算表的形式存储，表索引由原子谓词上的位向量赋值构成。表 2总结了确定有限自动机的规模。

评估与处理。 时序策略支持两种评估模式。在静态模式下，图 $\times$ 确定有限自动机积构造（第 4节，步骤 5a）会探索所有可达的乘积状态；任何乘积状态中的违规都意味着该属性可能在某条图路径上被违反。在运行时模式下，每个事件都会通过 $O(1)$ 表查找推进每条已编译规则对应的确定有限自动机状态。如果有任何规则进入违规状态，则配置的处理级别（警告、阻止、停止、升级）将决定最终决策。

表3：工作流语料库统计。

框架	工作流	平均节点数	平均边数
LangGraph	6	8.8	9.0
CrewAI	4	7.5	7.0
AutoGen	4	6.3	6.0
Google ADK	4	9.5	10.0
总计	**18**	**8.2**	**8.2**

表4：按类别划分的工作流语料库中发现的缺陷。

类别	缺陷类型	数量	严重程度	示例
结构性	死端节点	2	High	没有发送边的电子邮件草稿
	不可达出口	1	严重	无限循环，没有出口路径
	路由器形状违规	1	中等	辩论主持人，直接边
	缺少工具声明	1	Low	数据清理器，空工具集
策略	缺少人工关卡	10	High	未经审批的入职
	总计	**15**		

7 评估

拟议方法从六个方面进行评估：(i) 精选工作流语料库，(ii) 将结构性类别与策略类别分别考察的缺陷研究，(iii) 基于执行轨迹的时序策略评估，(iv) 在合成图上进行的可扩展性实验，(v) 与运行时护栏的比较，以及 (vi) 提取器准确度。

7.1 工作流语料库

本次评估采用一个精选基准，其中包含 18 个智能体工作流。这些工作流旨在体现各框架官方示例和教程中记录的常见模式与已知反模式。工作流使用 LangGraph、CrewAI、AutoGen 和 Google ADK，涵盖客户支持、RAG 流水线、代码生成、财务分析、合规审查、事件响应、招聘和营销等多个领域。由于该语料库的设计目标是纳入具有代表性的缺陷，因此下文的缺陷率反映的是工具在定向基准上的检测能力，而非生产系统中的基础发生率。我们计划开展一项更大规模的研究，分析从公开 GitHub 仓库中挖掘的工作流（第 9 节）。

表 3 总结了该语料库。图的规模为 5 至 12 个节点、4 至 14 条边，涵盖全部四种框架。节点类型分布体现了典型的智能体模式：大语言模型节点（推理步骤）和工具节点（外部 API 调用）占主导地位；在更复杂的工作流中，还会出现路由器、人工和子图节点。

7.2 缺陷研究

对每个语料库工作流都运行了全部六项结构检查，并设置为需要人工=True。表4按缺陷类别分别总结了发现。
区分了两类缺陷：

- **结构缺陷**（拓扑错误）：死端节点、不可达出口、活锁循环（可达但不存在通往终止状态的路径的节点）、路由器形状违规，以及缺少工具声明。无论运行环境如何，这些都是错误。该工具成功检测出了所有注入的结构缺陷：**18 个基准工作流中有 5 个**至少包含一项结构缺陷。反向可达性检查还将轮询式头脑风暴工作流中的循环节点识别为活锁：尽管这些节点可从入口到达，但无法到达任何出口节点。

- **策略违规**（可配置检查）：缺少人在回路关卡。此项检查取决于具体情境：在受监管领域（医疗保健、金融）中至关重要，但对于内部工具或低风险任务可能并非必要。**18 个基准工作流中有 10 个**在 require human=True 时缺少人工关卡；不过，标注数据显示，其中一些情况存在争议或是假正例——鉴于任务风险较低，这些工作流可能无需人工监督。

更值得关注的是，有一个工作流（轮询式头脑风暴智能体）存在不可达出口节点：智能体会无限循环，无法通过任何路径终止。另有两个工作流存在死端节点，导致执行悄无声息地停止。仅靠测试很难发现这些结构性缺陷，因为它们只会在特定执行路径上显现。

对于每项失败检查，验证器都会生成一条见证轨迹。例如，电子邮件分流工作流中的死端会产生如下见证路径：start $\rightarrow$ classify $\rightarrow$ router $\rightarrow$ ~~n~~ormal handler$\rightarrow$ draft response（卡住）。

静态时序验证。图 $\times$ DFA 积构造应用于 18 个工作流中的全部 15 条已编译时序策略。静态分析正确识别了运行时轨迹评估也检测到的所有禁用工具违规和蕴含未来违规，此外还标记出一项违规（起草邮件后缺少人工审核步骤）；在该工作流中，随机轨迹恰好避开了违规路径。所有情况下，乘积状态探索均低于 $|V| \cdot |Q_{\text{DFA}}|$，乘积状态最多为 24 个（12 节点图，2 状态 DFA）。

图 5 展示了各框架中的缺陷分布，图6 展示了各检查类型的精确率细分。所有结构检查的精确率均为满分（$P = 1.0$）；人工存在性检查的精确率为 $P = 0.75$，这是因为在一个低风险工作流中出现了一次假阳性，而在该工作流中，人工监督或许并无必要。

7.3 时序策略评估

为评估时序 DSL，我们根据涵盖六个领域的现实需求，定义了 15 条具体安全策略：安全性、通信、隐私、开发运维、数据治理和合规。表 5 列出了一些代表性示例。

全部 15 项策略均成功编译为确定有限自动机。这些策略涵盖了全部七种领域专用语言表达式形式：3 项禁止策略、5 项未来蕴含模式、2 项有界响应、1 条响应链、1 项直到模式、2 项合取和 1 项析取。没有任何策略需要超出七形式片段的完整线性时序逻辑表达能力。
每个工作流生成了 10 条随机执行轨迹（共 180 条），并针对每条轨迹评估每项策略。评估流程通过检查策略的原子谓词是否出现在工作流的执行轨迹中，确定策略适用性。

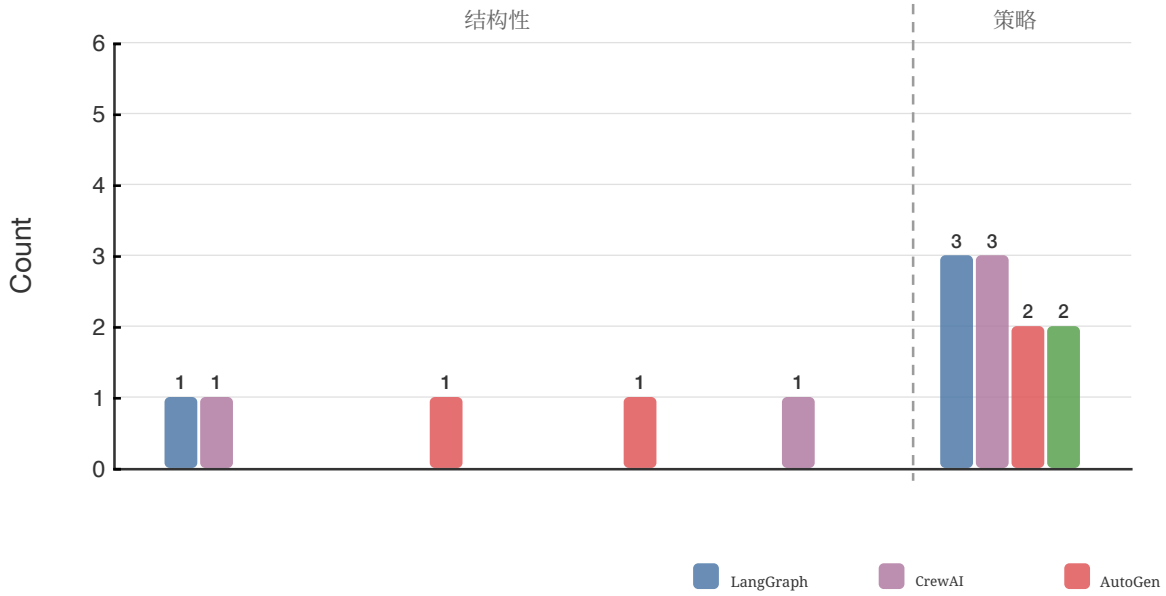


图5：按缺陷类型分组的各框架缺陷分布。虚线将结构缺陷（左侧）与策略违规（右侧）分隔开。

表5：评估的代表性时序策略。

策略	DSL 表达式	领域
无破坏性操作	G !工具:drop table	安全性
电子邮件审查	工具:起草 email -> F 工具:人工审查	通信
个人身份信息匿名化	工具:获取个人身份信息 -> F[<=3] 工具:匿名化	隐私
部署审批	工具:部署 ->F 工具:批准	开发运维
起草-审查-发送	工具：起草 -> F 工具：审查 -> F 工具：发送	通信

领域特定语言范围论证。 定义 15项策略后，按照各自使用的领域特定语言形式对其进行分类。全部15项策略均属于七形式片段；没有任何策略需要嵌套时态算子、过去时模态或超出语法范围的构造。

虽然这些政策由作者制定，但其依据来自已发布的监管与行业要求：支付需要人工依据《通用数据保护条例》第22条（自动化决策需要人工审查） [14]；个人身份信息匿名化有界的依据《通用数据保护条例》的数据最小化原则；部署需要审批依据 SOC 2 变更管理控制 [5]；电子邮件需要审查依据组织沟通政策；不得执行破坏性操作依据 OWASP 大语言模型十大风险（LLM06：过度代理） [24]；并且要么记录日志，要么进行审计依据 SOC 2 审计日志要求。这些政策以外部标准为依据，说明该领域特定语言片段涵盖了实际的安全需求，不过，若能对生产环境部署进行系统的需求调查，则可提供更有力的证据。

需要承认的是，来自单个研究团队的15项策略并不构成一项详尽的需求调查。这些策略经过有意筛选，涵盖六个不同领域，并确保所有七种领域特定语言形式都得到应用。

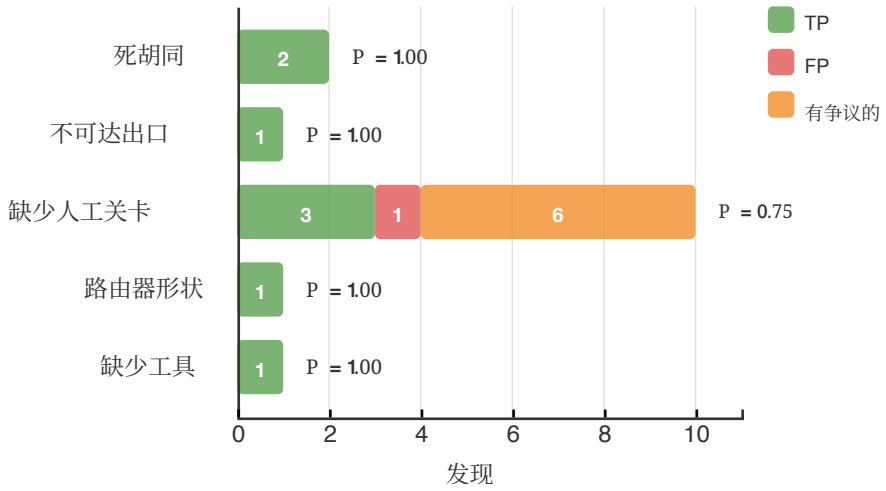


图6：每项结构检查的精确率。每个柱状条显示真正例（TP）、假正例（FP）和有争议的发现的构成。精确率 P 计算为 TP/(TP + FP)。

表 6：结构检查时间与图规模的关系。监视器编译和评估与规模无关。

节点	边	结构（毫秒）	监测器评估（毫秒）
50	104	0.04	
100	214	0.07	
200	430	0.22	
500	1,074	1.24	4.0
1,000	2,128	4.02	
2,000	4,252	21.2	
5,000	10,713	104.7	

7.4 可扩展性

合成图以七种规模（50–5,000个节点）生成，边密度为2.0，并测量结构检查时间（10次试验的中位数）。图7 和表 6报告了结果。

对于最多 1,000 个节点的图，结构检查在 $|V| + |E|$ 中大致呈线性扩展；在更大规模下，由于路由器形状检查和工具声明检查中的二次边扫描，增长呈超线性。即使在 5,000 个节点时（远大于实践中遇到的任何生产智能体工作流），验证也能在 $\sim$105 毫秒内完成。时序监视器编译（5 条规则为 0.065 毫秒）和评估（1,000 个事件为 4.0 毫秒， $\sim$247,000 个事件/秒）均不受图规模影响。

目前真实的智能体工作流规模较小（语料库中为 5–12 个节点），因此可扩展性实验展示的是无上限性，而非实际必要性。随着智能体系统发展为多智能体组合和层次化子图结构，更大的图将变得更加常见。在本次评估中，尚未测量将所有条件边视为图中可行边 $\times$ 确定有限自动机积构造所导致的实际假阳性率，这也是未来量化研究的一个方向。

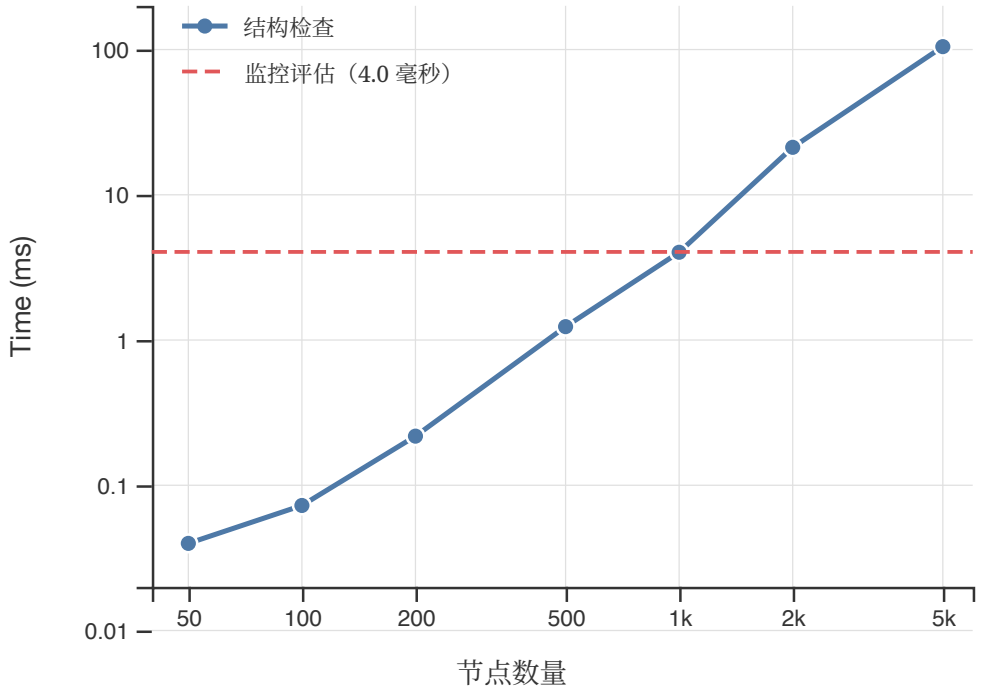


图 7：结构检查时间与图规模的关系（双对数坐标，10 次试验的中位数）。对于最多包含 5,000 个节点的图，验证耗时仍不到一秒。

与临时脚本的比较。 有人可能会问，使用一个简单的 Python 脚本遍历图的邻接表，是否也能检测出相同的缺陷。对于单项检查（例如查找没有出边的节点），答案是肯定的——图算法是标准的。Agentproof 的价值不在于算法创新，而在于：（i）从四种异构框架 API 中自动提取信息；（ii）通过统一类型系统使检查能够跨框架移植；（iii）生成见证轨迹，精确定位存在缺陷的路径；以及（iv）提供时序策略层，将安全规范与验证机制分离。定制脚本需要针对每个框架重新编写，并且不具备可组合策略评估能力。

7.5 跨框架案例研究

为验证 Agentproof 的跨框架能力，我们从不同框架中选取了三个具有代表性的工作流：LangGraph 中的事件响应工作流、Google ADK 中的合规审查流程，以及 AutoGen 中的变更控制流程。表 7 报告了图统计和提取器运行时间；表 8 汇总了代表性轨迹（每个工作流包含一条安全轨迹和两条不安全轨迹）上的结构检查结果和时序监控结果。

7.6 与运行时护栏的比较

表 9 对静态方法与具有代表性的运行时护栏工具进行了定性比较。

场景 A：一个 u　在极少触发的条件分支中存在无法到达的出口节点。Agentproof 的静态　c

表7：代表性工作流的图统计和提取器运行时间。节点类型列报告
TOOL/LLM/ROUTER/HUMAN/SUBGRAPH 节点的数量；其他类型（EN-TRY/出口/PASSTHROUGH）包含在 $|V|$ 中。

研究	$\|V\|$	$\|E\|$	Tool	LLM	路由器	人工	Subg	Dir	Cond	Par	Loop	中位数（毫秒）
LangGraph（事件响应）	8	8	2	2	1	1	0	6	2	0	0	0.186
ADK（合规）	13	16	4	3	0	1	3	12	0	3	1	0.019
AutoGen（变更控制）	6	5	0	3	0	1	0	5	0	0	0	0.007

表 8：各案例研究的代表性轨迹（一个安全轨迹和两个不安全轨迹）上的结构检查结果和时序监控结果。

研究	结构性（通过/总数）	正常路径	禁止工具	策略违规
LangGraph（事件响应）	5/5	PASS	HALT	升级
ADK（合规）	5/5	PASS	HALT	已阻止
AutoGen（变更控制）	5/5	PASS	HALT	HALT

分析会立即检测到这一问题；除非在测试期间执行该分支，否则运行时护栏会漏掉它。

场景 B： 大语言模型在回复中生成有害内容。运行时内容过滤器（LlamaGuard、NeMo Guardrails）可以检测到此类内容；Agentproof 无法检测，因为大语言模型输出语义不在其范围内。
场景 C： 一条条件路径绕过了必需的人工审批关卡。Agentproof 的人工关卡覆盖检查会标记从入口到敏感工具的未受保护路径；如果未触发绕过路径，运行时工具可能无法发现它。

这两种方法相辅相成：静态验证能够全面捕获拓扑层面缺陷，而运行时护栏则处理内容层面和依赖上下文的违规。

7.7 提取器准确度

提取器保真度通过测试套件中的框架特定示例进行了验证：包括 5 个 LangGraph、7 个 CrewAI、7 个 AutoGen 和 8 个 ADK 测试用例（共 27 个）。对于每个提取器，都将提取的图与人工标注的真实标注进行比较，并测量节点检测精确率/召回率、节点类型分类准确率以及边检测精确率/召回率。

人工节点检测局限。 分类错误的主要来源是 LangGraph 中的人工节点检测，该检测依赖命名启发式（名称中包含“human”）：不遵循这一惯例的节点会被误分类为大语言模型节点。以下按可靠性递增的顺序列出四种人工节点检测策略：(1) 命名启发式（当前默认策略）：将节点名称与关键词进行匹配——实现成本低，但较为脆弱。(2) 中断注解：检测 LangGraph 的中断前／中断后标记——依赖特定框架，但语义精确。(3) 输入调用检测：通过抽象语法树检查，在节点函数中识别 input() 或等效的阻塞调用——可跨框架使用，但可能产生假正例。(4) 显式类型注解：要求开发者通过装饰器或元数据字段标记人工节点——可靠性最高，但需要开发者采用。在测试套件中，只有策略 (1) 会导致误分类；策略 (2)–(4) 仍需在更大规模的真实世界语料库上验证。策略 (2) 被列为下一个版本的优先选项，因为它无需开发者采取任何操作。

表9：静态验证与运行时护栏。

属性	Agentproof（静态）	运行时工具
检测时间	Pre-部署	在执行时
结构缺陷	✓	路径-依赖
有害大语言模型输出		✓
运行时开销	None	每次调用 la-迟
覆盖率	所有路径	已执行路径
见证轨迹	✓	堆栈跟踪

提取结果。 考虑到上述人工节点注意事项，在测试套件中，所有框架的节点检测精确率和召回率均达到 100%。入口/出口哨兵和工具节点（通过工具绑定检测）的节点类型分类准确率为 100%；AutoGen 代理类型（通过类层次结构检测）的节点类型分类准确率也为 100%。在所有受测框架中，边检测的精确率和召回率均为 100%。

在使用程序化框架桩的测试套件中，提取器对节点和边的精确率与召回率均达到完美水平。这验证了提取逻辑能够处理已知的 API 模式，但并未衡量其面对现实世界中多样化工作流定义时的稳健性。基于抽象语法树的提取器（scripts/ast extractor.py）提供了一种独立的提取路径；若能在相同工作流上验证运行时提取与抽象语法树提取结果的一致性，将增强可信度，这也是未来交叉验证的一个方向。

8 相关工作

本节将本文置于四个研究领域之中。运行时工具能够捕捉内容层违规，但会遗漏结构缺陷。通用模型检查器可以验证相同的性质，但需要手动建模。时序监控已得到充分研究，但尚未应用于智能体工作流。智能体安全研究关注对齐，而非编排拓扑。Agentproof 填补了运行时内容检查与结构验证之间的空白。

8.1 运行时智能体安全工具

多个工具在大语言模型输出或工具调用发生时实施安全防护。NVIDIA NeMo Guardrails [23] 在大语言模型与工具调用之间插入可编程对话护栏，支持主题控制和输出过滤。Guardrails AI [16] 通过验证器封装大语言模型输出，检查其格式、毒性和事实一致性。LlamaGuard [18]使用微调分类器检测模型输出中的不安全内容。Rebuff [26]专门关注提示注入检测。这些工具在运行时运行，仅当违规代码路径实际执行时才能捕获违规。Agentproof 则具有互补作用：它在部署之前验证工作流拓扑的结构属性，捕获不可达出口、死端分支和缺少人工关卡等缺陷，而运行时工具只有在测试期间触发缺陷路径时才能检测到这些问题。

表10：与通用模型检查器的比较。

Tool	输入	性质	建模	Time	领域
SPIN	Promela（手动）	完整线性时序逻辑	High	快速（小型）	None
NuSMV	SMV（手动）	计算树逻辑 + 线性时序逻辑	High	快速（小型）	None
CBMC	C 源代码	断言	中等	有界的	None
Agentproof	自动提取	安全 LTL 片段	**None**	$O(\|V\|+\|E\|)$	**智能体工作流**

SPIN、NuSMV 和 CBMC 都需要手动将智能体工作流转换为各自的输入语言。Agentproof 可从框架 API 自动提取模型，从而完全省去建模步骤。

8.2 静态分析与模型检验

经典模型检验 [11] 使用 SPIN 等工具验证有限状态系统是否符合时序规约， [17], NuSMV [10], 和 CBMC [20]。抽象解释 [12] 为数据流分析提供可靠的过近似。

表10 给出了定性比较。关键区别在于**建模工作量**：SPIN 要求将工作流手动转换为 Promela，将 NuSMV 转换为 SMV，并将 CBMC 转换为带注释的 C。对于语料库中的电子邮件分流工作流（8 个节点、7 条边），等效的 Promela 模型需要 ∼60 行代码，并且需要手动指定状态空间、状态转移和 LTL 性质（见工件中的`corpus/comparisons/email_triage.pml`）。Agentproof 通过一次函数调用即可自动提取相同的模型。

Agentproof 将模型检测思想应用于一个新领域：从编排框架 API 中提取的智能体工作流图。其贡献不在于图算法本身（这些算法是标准的），而在于：（i）从异构框架表示中自动提取；（ii）针对智能体安全定制的领域专用性质语言；以及（iii）通过实证证明智能体工作流包含可由这些方法捕获的结构缺陷。

8.3 时序逻辑监控

运行时验证（RV）社区已经开发出复杂的监控框架。JavaMOP [19] 根据多种形式化语言编写的规范监控 Java 程序。Bauer 等人 [9] 提出了三值 LTL 监控，用于区分“当前满足”和“永久满足”。Barringer 等人 [8] 提出了用于基于规则的监控的 EAGLE 框架。这些时序监控器经过有意的轻量化设计，面向 LTL 的安全片段，每条规则使用具有 2–$O(k)$ 个状态的确定有限自动机，以牺牲表达能力换取部署简便性。在评估中，全部 15 项实际智能体安全策略都属于这一片段，从实证上验证了这一设计选择（第7.3节）。

8.4 智能体架构与安全性

近期关于人工智能智能体安全的研究涵盖多个维度。Anthropic 的负责任扩展政策 [6]概述了安全承诺，其中包括针对自主能力的评估阈值。METR [21]提供了对自主智能体能力及其风险的评估。关于多智能体协调安全 [15] 的研究考察了智能体间交互的风险，包括非预期的涌现行为。这些工作聚焦于对齐、能力评估和政策治理。本文则探讨一个互补层面：**编排图本身的结构性健全性**。即使是对齐良好的大语言模型，如果工作流图将其引导至非预期路径，或绕过了必要的审批步骤，也可能产生不安全的结果；

从而经过非预期路径或绕过必要的审批步骤。

8.5 业务流程验证

业务流程管理（BPM）领域对工作流健全性进行了广泛研究。Van der Aalst 关于工作流网的基于 Petri 网的健全性检查 [29] 的工作确保每个案例都能完成，且不存在死变迁；这些性质与 Agentproof 的出口可达性和死端检查十分相似。BPMN 验证工具 [13] 也会对行业标准流程模型应用类似的分析。这项工作的领域有所不同：智能体工作流图包含带类型的节点（大语言模型、工具、人工），其语义因框架而异，BPM 工具无法对其建模；此外，抽取流水线的目标是程序化 API 对象，而非可视化流程模型。

8.6 受约束解码与结构化生成

受约束解码在令牌级别强制规定输出结构：Guidance [22] 和 Outlines [28]会将语法或 JSON 模式编译为令牌掩码，从而确保输出格式良好。这些技术在单次大语言模型调用中发挥作用，限制模型能够生成的内容。Agentproof 作用于不同的粒度：它约束的是计算步骤（节点）之间的状态转移，而非任何单个步骤中生成的内容。这两种方法互为补充。

9 局限性与未来工作

工作流结构与大语言模型语义。 拟议方法验证工作流拓扑 以及执行所发出的事件流的性质。它不证明大语言模型输出的语义属性（例如事实性或意图），这些属性取决于非确定性的模型行为以及提示词／上下文选择。

提取器启发式方法。 框架发展迅速，不同版本会暴露不同的内部表示。提取器采用启发式方法（例如，根据命名约定识别人工审核节点），这些方法未必适用于所有代码库。目前的验证使用了 27 个基于桩的测试用例（第 7.7 节）；基于桩的验证与真实世界工作流定义的多样性之间的差距尚未测量。基于 AST 的提取器（`scripts/ast extractor.py`）提供了一条独立的提取路径，可作为交叉验证工具，以增强对提取保真度的信心。

运行时图变更（T3）。 一些框架允许在执行期间动态修改工作流图。如果变更后的图未经重新验证，静态保证便不再成立。将重新验证钩子集成到框架事件系统中，是一种自然的扩展方向。

时序 DSL 的表达能力。 该 DSL 涵盖七种表达式形式，面向 LTL 的安全片段，但不支持完整的 LTL、超过两层的嵌套时态算子、过去时模态，或超出步数计数范围的实时界限。全部 15 项评估策略均符合这一片段；然而，这些策略由设计该 DSL 的同一团队编写，因此存在自我选择偏差：不符合语法的需求不太可能被提出。这些策略的灵感来自外部标准（通用数据保护条例 [14], SOC 2 [5], OWASP 大语言模型

顶端 10 [24]），这在一定程度上缓解了这一问题，但仍需对独立生产环境部署中的安全需求进行系统调查，以验证 DSL 是否足够。扩展至更丰富的时序片段（例如，过去时 LTL、实时约束）是未来的工作。

假正例管理。某些被标记的缺陷可能是有意的设计选择（例如，将死端节点用作有意触发错误并停止的状态，或者在低风险任务的工作流中有意省略人工关卡）。未来的工作包括基于注释的抑制（例如，# agentproof: ignore dead-end 注释）、严重性等级以及可配置的验证配置文件。该库已支持用于排除节点级问题的 suppressions 参数。

静态时间保守性。图 × 确定有限自动机乘积构造会探索所有拓扑路径，包括因路由器条件或大语言模型决策逻辑而在运行时可能不可行的路径。这使分析具有可靠性（不会产生假阴性），但也可能较为保守（产生假正例）。未来的一个研究方向是在乘积构造中纳入边条件，以剪除不可行路径。

语料库规模。评估使用了 18 个由作者构建的工作流。有效性威胁：由于语料库旨在纳入具有代表性的缺陷和反模式，观察到的缺陷率反映的是在定向基准上的检测能力，而非生产系统中的普遍情况。要将这些比率推广到更广泛的场景，需要在独立编写的工作流上进行验证。Agentproof 提供了 GitHub 挖掘流水线（scripts/scrape workflows.py）和基于抽象语法树的备用提取器（scripts/ast extractor.py），可直接用于构建更大规模的真实语料库。还需要对数百个开源代码库开展大规模研究，以确定基准发生率，并为社区提供可复用的基准。

10 结论

智能体框架已经将行为编码为显式的工作流图。Agentproof 利用这一结构，无需增加运行时开销即可实现安全属性的部署前验证。
Agentproof 提供统一的图模型，包含六项结构检查和见证轨迹；提供覆盖 LTL 安全片段的时序策略领域专用语言，支持静态和运行时两种评估模式；还为四种主流智能体框架提供自动提取器，从而免去了通用模型检查器所需的手动建模工作。在包含 18 个工作流的精选基准中，5 个存在结构缺陷（死胡同、不可达出口），10 个在执行策略时缺少人工关卡。评估的全部 15 项时序安全策略均可用七形式 DSL 片段表示；即使图包含 5,000 个节点，验证也能在不到一秒内完成。

主要的技术贡献并非图算法本身——这些算法早已成熟——而在于：(i) 发现了一个适合直接应用这些算法的实用新领域；(ii) 开发了提取器，将四种异构框架 API 规范化为统一的可分析表示；(iii) 通过实证发现，真实世界的智能体工作流模式中存在可用这些方法检测的结构缺陷。

静态验证并不能取代运行时护栏；这两种方法相辅相成。通过在部署前全面捕获拓扑层面缺陷，静态分析减少了运行时强制执行需要覆盖的安全范围，从而提升两个层面的有效性。

工件可用性。Agentproof 工具、精选语料库、时序策略和所有评估脚本均可在以下网址获取：https://github.com/NordicAgents/AgentProof 并采用 MIT 许可证。可复现性脚本（`scripts/reproduce all.sh`）可运行完整的评估流水线。

参考文献

[1] AutoGen AgentChat。Python 软件包 autogen-agentchat（微软），2025年。访问日期：2026年2月28日。[2] CrewAI 文档。https://docs.crewai.com，2025年。访问日期：2026年2月28日。[3]Google 智能体开发套件（ADK）文档。https://google.github.io/ adk-docs/，2025年。访问日期：2026年2月28日。[4]LangGraph 文档。https://docs.langchain.com/oss/python/langgraph/ overview，2025年。访问日期：2026年2月28日。[5] 美国注册会计师协会。SOC 2——信任服务标准。https://www.aicpa.org/topic/audit-assurance/ audit-and-assurance-greater-than-soc-2，2017年。访问日期：2026年3月1日。[6]Anthropic。Anthropic 的负责任扩展政策。https://www.anthropic.com/ responsible-scaling-policy，2023年。访问日期：2026年3月1日。[7] Christel Baier 和 Joost-Pieter Katoen。《模型检验原理》。麻省理工学院出版社，2008年。[8] Howard Barringer、Allen Goldberg、Klaus Havelund 和 Koushik Sen。基于规则的运行时验证。载于：国际验证、模型检验与抽象解释会议（VMCAI）。施普林格，2004年。[9] Andreas Bauer、Martin Leucker 和 Christian Schallhart。针对线性时序逻辑和 TLTL 的运行时验证。《ACM软件工程与方法学汇刊》，20(4)，2011年。[10] Alessandro Cimatti、Edmund Clarke、Enrico Giunchiglia 等。NuSMV 2：一种用于符号模型检验的开源工具。载于：国际计算机辅助验证会议（CAV）。施普林格，2002年。[11] Edmund M. Clarke、Orna Grumberg 和 Doron A. Peled。《模型检验》。麻省理工学院出版社，1999年。[12] Patrick Cousot 和 Radhia Cousot。抽象解释：一种用于程序静态分析的统一格模型。载于：第4届ACM编程语言原理研讨会（POPL），1977年。[13] Remco M. Dijkman、Marlon Dumas 和 Chun Ouyang。BPMN 业务流程模型的语义与分析。《信息与软件技术》，50(12):1281–1294，2008年。[14] 欧洲议会和欧盟理事会。《欧盟条例2016/679》（通用数据保护条例），第22条：自动化个人决策，包括画像分析。https://gdpr-info. eu/art-22-gdpr/，2016年。访问日期：2026年3月1日。[15] Iason Gabriel 等。高级人工智能助手的伦理。arXiv 预印本 arXiv:2404.16244，2024年。

[16] Guardrails AI。Guardrails AI：为大语言模型添加防护机制。https://github.com/guardrails-ai/guardrails，2023年。访问日期：2026年3月1日。[17] Gerard J. Holzmann。模型检验器 SPIN。《IEEE 软件工程汇刊》，23(5):279–295，1997年。[18] Hakan Inan、Kartikeya Upasani、Jianfeng Chi 等。LlamaGuard：用于人类与人工智能对话的基于大语言模型的输入输出防护。arXiv 预印本 arXiv:2312.06674，2023年。[19] Dongyun Jin、Patrick O'Neil Meredith、Choonghwan Lee 和 Grigore Ro˛su。JavaMOP：高效的参数化运行时监控框架。载于：国际软件工程会议（ICSE），2012年。[20] Daniel Kroening 和 Michael Tautschnig。CBMC——C 有界模型检查器。载于：国际系统构造与分析工具和算法会议（TACAS）。施普林格，2014年。[21] METR。METR：模型评估与威胁研究。https://metr.org，2024年。访问日期：2026年3月1日。[22] 微软。Guidance：用于控制大语言模型的引导语言。https: //github.com/guidance-ai/guidance，2023年。[23] 英伟达。NeMo Guardrails：用于可控、安全的大语言模型应用的工具包。https: //github.com/NVIDIA/NeMo-Guardrails，2023年。访问日期：2026年3月1日。[24] OWASP 基金会。大语言模型应用 OWASP 十大风险。https://owasp.org/www-project-top-10-for-large-language-model-applications/，2025年。2.0 版。访问日期：2026年3月1日。[25] Amir Pnueli。程序的时序逻辑。载于：第18届计算机科学基础年度研讨会（FOCS），1977年。[26] Protectai。Rebuff：自我强化型提示注入检测器。https://github.com/protectai/ rebuff，2023年。访问日期：2026年3月1日。[27] Timo Schick、Jane Dwivedi-Yu、Roberto Dess`ı、Roberta Raileanu、Maria Lomeli、Eric Hambro、Luke Zettlemoyer、Nicola Cancedda 和 Thomas Scialom。Toolformer：语言模型可以自学使用工具。载于：神经信息处理系统进展（NeurIPS），第36卷，2023年。[28] .txt。Outlines：结构化文本生成。https://github.com/outlines-dev/outlines，2023年。[29] Wil M. P. van der Aalst。《流程挖掘：业务流程的发现、一致性与增强》。施普林格，2011年。[30] Moshe Y. Vardi。使用双向自动机推理过去。载于：国际自动机、语言与编程研讨会（ICALP），1994年。[31] Lei Wang、Chen Ma、Xueyang Feng 等。基于大语言模型的自主智能体综述。《计算机科学前沿》，18(6)，2024年。

[32] Qingyun Wu、Gagan Bansal、Jieyu Zhang、Yiran Wu、Beibin Li、Erkang Zhu、Li Jiang、Xiaoyun Zhang、Shaokun Zhang、Jiale Liu 等。AutoGen：通过多智能体对话赋能新一代大语言模型应用。发表于语言建模会议（*COLM*），2024年。

[33] Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik Narasimhan 和 Yuan Cao。ReAct：在语言模型中协同推理与行动。发表于国际学习表征会议（*ICLR*），2023年。

A 可靠性证明

本附录证明每项结构检查都是可靠的：如果检查通过，则相应的安全性质对所有有效执行轨迹均成立。

引理 1 (退出可达性可靠性**)。** 如果$\mathrm{ExitReach}(G)$ 成立（即从 v_0 开始的广度优先搜索／深度优先搜索访问了 V_f中的所有节点），那么对于每个出口节点 $v_f \in V_f$，在 G中都存在一条从 v_0 到 v_f 的有向路径。

证明。 该检查通过从 v_0进行广度优先搜索，计算 Reach(v_0，从而正确识别通过有向边 [11]可达的所有节点。若 $v_f \in \mathrm{Reach}(v_0)$，则根据广度优先搜索的正确性，存在一条路径$v_0 \to \cdots \to v_f$。由于仅当 $V_f \subseteq \mathrm{Reach}(v_0$ 时检查才会通过，因此结论成立。 □

引理 2 (死端可靠性**)。** 如果$\mathrm{NoDead}(G)$ 成立，那么G 的每条轨迹中的每个非出口节点都至少有一个后继节点，因此没有极大轨迹会在非出口节点处终止。

证明。 令 v 为任意满足 $\kappa_V(v) \neq \textsc{exit}$ 的节点。由于 NoDead(G) 成立，存在 u 使得 $(v, u) \in E$。因此，如果一条轨迹 π 经过 v，就可以通过 u延长该轨迹。只有当轨迹终止于 V_f 中的一个节点，或终止于没有出边的节点时，它才是极大的（即无法延长）。由于每个非出口节点都有出边，所有极大轨迹都终止于出口节点。

□

引理 3 (路由器形状可靠性**)。** 如果$\mathrm{RouterShape}(G)$ 成立，则路由器节点的每个出射转换都明确标记为条件转换。

证明。 这直接来自谓词定义：该检查会遍历所有边 $(v, u) \in E$，其中 $\kappa_V(v) = \textsc{router}$，并验证 $\kappa_E(v, u) = \textsc{conditional}$。通过检查即可确立全称量化。

□

引理 4 (人工关卡可靠性**)。** 如果$\mathrm{HumanGate}(G)$ 成立，则图中至少包含一个被归类为人在回路步骤的节点。

证明。 该检查会搜索满足 $\kappa_V(v) = \textsc{human}$的 $v \in V$。通过检查即可证明其存在。 □

引理 5 (工具声明可靠性**)。** 如果$\mathrm{ToolDecl}(G)$ 成立，则每个工具节点都会明确声明其工具集。

证明。 对于每个满足 $\kappa_V(v) = \textsc{tool}$的 v，该检查都会验证 $T(v) \neq \emptyset$。通过检查即可确立该全称命题。

□

引理 6 (反向可达性可靠性**)。** 如果$\mathrm{ExitReachAll}(G)$ 成立（即$\mathrm{Reach}(v_0) \subseteq \mathrm{RevReach}(V_f)$），那么对于每个从 v_0 可达的节点 v，都存在一条从 v 到某个出口节点 $v_f \in V_f$ 的有向路径。

证明。 该检查从 V_f 中的所有节点出发，在反向邻接上通过广度优先搜索计算 RevReach(V_f)。根据广度优先搜索的正确性， $v \in$RevReach(V_f) 意味着存在一条路径 $v \to \cdots \to v_f$，满足某个 $v_f \in V_f$。只有当 Reach(v_0) $\subseteq$RevReach(V_f) 时，该检查才会通过，因此结论对每个可达节点都成立。 □

引理 7 (人类门覆盖可靠性)。 如果对于一组 S 敏感工具名称，$\mathrm{HumanGateCov}(G, S)$ 成立，那么从 v_0 到调用 S 中某个工具的节点的每条路径，都至少经过一个 HUMAN节点。

证明。 该检查通过移除所有带 HUMAN 类型的节点及其关联边，构造修改后的图 G'，然后计算 Reach$_{G'}$(v_0)。如果 Reach$_{G'}$(v_0) 中没有敏感工具节点，那么在原图 G 中，从 v_0 到此类节点的每条路径都必须经过一个被移除的 HUMAN 节点。反之，如果在 G' 中可以到达敏感工具节点，则检查失败，并报告这条无人工路径。 □

引理 8 (静态时序可靠性)。 对于一个带有确定有限自动机$(Q, q_0, \delta, F_{\text{viol}})$ 的编译后监控规则以及智能体工作流图 G，如果图 $\times$ 与确定有限自动机的积构造未报告违规，则 G 的任何执行轨迹都不会违反该时序性质。

证明。 乘积图广度优先搜索会探索 $V \times Q$ 中所有可达状态。每个乘积状态 (v, q) 表示当前位于图节点 v，且确定有限自动机处于状态 q。对于 G 中 v 的每个后继节点 u，确定有限自动机通过 $q' = \delta(q, \sigma(v))$ 转移，其中 $\sigma(v)$ 是节点 v 的事件符号，并将乘积后继节点 (u, q') 加入队列。如果没有可达乘积状态包含 $q' \in F_{\text{viol}}$，那么任何执行轨迹——即经过乘积状态的路径——都无法到达违规状态。这是一种过近似：该乘积会探索所有图路径，这是实际运行时轨迹的超集（因为运行时轨迹取决于路由器节点处大语言模型的决策）。因此，该分析是可靠的，但可能较为保守。 □

引理 9 (时序监测器健全性)。 对于三种基础确定性有限自动机模式（禁止、未来蕴含模式、直到模式）中的每一种，如果编译后的监测器在有限轨迹 π 上报告未发现违规，那么 π满足相应的时序公式。

证明。 证明分情况进行。
禁止（$\mathbf{G}\,\neg a$）：确定有限自动机有两个状态： s_0 （安全）和 s_1 （已违规，吸收态）。转移函数将 $s_0 \xrightarrow{a} s_1$ 和 $s_0 \xrightarrow{\neg a} s_0$映射为相应状态，其中 s_1 为吸收态。如果监测器从未进入 s_1，那么 a 在每一步都为假，从而证明 $\mathbf{G}\,\neg a$。
未来蕴含模式（$a \to \mathbf{F}\,b$）：确定有限自动机有三个状态： s_0 （空闲）、 s_1 （等待 b）和 s_2（已违规，吸收态）。状态转移： $s_0 \xrightarrow{a \wedge \neg b} s_1$； $s_1 \xrightarrow{b} s_0$； $s_1 \xrightarrow{a \wedge \neg b} s_2$。如果监测器从未进入 s_2，那么每次出现 a 之后，都会在 a再次出现之前出现 b 。
直到模式 ($a\,\mathbf{U}\,b$)*:*确定有限自动机有三个状态： s_0（等待中）、 s_1（已满足，吸收态）、 s_2（已违反，吸收态）。状态转移： $s_0 \xrightarrow{b} s_1$； $s_0 \xrightarrow{a \wedge \neg b} s_0$； $s_0 \xrightarrow{\neg a \wedge \neg b} s_2$。如果监测器从未进入 s_2，那么 a在每一步都成立，直到 b发生，从而证明 $a\mathbf{U}\ b$。
在每种情况下，确定有限自动机的状态不变式都通过对轨迹前缀进行归纳法得以维持。 □

B 时序 DSL 语法

时序策略领域专用语言的完整巴科斯-诺尔范式语法：

<规则> ::= <禁止>| <蕴含_未来>| <直到模式>| <有界的>| <链>| <合取>| <析取><禁止> ::="G" "!" <原子命题><蕴含_未来>::= <原子命题> "->" "F" <原子命题><直到模式> ::= <原子命题> "U" <原子命题><有界的> ::= <原子命题> "->" "F[<=" <整数> "]" <原子命题><链> ::= <原子命题> ("->" "F" <原子命题>){2,}<合取>::="(" <规则> ")" "AND" "(" <规则> ")"<析取>::="(" <规则> ")" "OR" "(" <规则> ")"<原子命题> ::="工具:" <名称>| "操作:" <名称>|"决策:" <名称>| <标签_名称>

<名称> ::= [a-zA-Z_][a-zA-Z0-9_]*<标签_名称> ::= [a-zA-Z_][a-zA-Z0-9_]*<整数> ::= [1-9][0-9]*

谓词匹配。 原子谓词与事件字典进行匹配：当事件的工具名称字段等于 X 时，tool:X 匹配；action:X 匹配动作类型；decision:X 匹配决策字段；单独的名称则匹配事件标签集合中的成员。