arXiv:2603.06710v1 [cs.LO] 5 Mar 2026

超越布尔值的挖掘：学习数据变换与时序规范

Sam NicholasKouteili[1],William Fishell[2], Christian Scaff[2], Mark Santolucito[2†], 以及 Ruzica Piskac[1]

[1] 耶鲁大学[2] 哥伦比亚大学（† 巴纳德学院，哥伦比亚大学）

摘要。 从执行轨迹中挖掘规范，为自动捕获具有特征的系统行为提供了一种方式。然而，现有方法大多局限于对事件进行布尔抽象，从而限制了其表达数据感知属性的能力。本文将挖掘过程扩展到更丰富的数据类型。首先，我们利用语法引导综合（SyGuS）技术，在所研究的领域中建立能够覆盖轨迹集合的候选函数。为了在时间维度上刻画这些函数应用，我们形式化了 TSL$_f$ 的语义：TSL 是时序流逻辑（TSL）的一种有限前缀解释，并在 LTL$_f$ 的基础上支持一阶谓词和函数更新。由此，我们得以统一一个用于学习系统数据变换与时序规范的相应过程。我们在 OpenAI-Gymnasium ToyText 环境中，根据挖掘出的规范综合反应式程序，展示了我们的方法；结果表明，与被动学习基线相比，在推广问题实例上，我们的方法更加稳健，样本效率高出数个数量级。

1 引言

规范挖掘是从演示中推导逻辑属性的过程，旨在发现能够统一观测或区分正反例的不变式 [3, 34, 36]。手动编写规约容易出错，许多案例表明，有缺陷的规约曾导致严重的软件故障。例如，Linux 内核中的错误规约造成了可被利用的漏洞 [20], ，而规约缺漏则引发了大规模云系统中的严重故障 [37]。为减少对人类直觉的依赖，规范挖掘可直接从执行中自动学习。

规范挖掘的标准方法会分析执行轨迹，这些轨迹记录了系统随时间变化的行为。因此，时序逻辑自然而然地被用作表达规约的形式体系。线性时序逻辑（LTL）[38, 16]及其有限前缀解释 LTL$_f$ [19] 长期以来一直是编码系统时序性质的标准，并应用于基于智能体的规划 [5, 2] 和运行时验证 [17]。Scarlet [40] 和 Bolt [8] 等近期挖掘工具展示了挖掘 LTL$_f$ 规约的可行性。

尽管 LTL 和 LTL$_f$ 已得到广泛应用，但它们都是命题形式系统，这意味着所有事件都被编码为布尔值。这会带来多项挑战。例如，要表达依赖于变量演化的性质（即“移动 x

到最近障碍物上方的单元格”），我们可以手动构造谓词及其解释（需要人工干预），也可以对轨迹数据进行位爆破（导致公式大小增加，并可能引入虚假的语义关系）。时序流逻辑（TSL） [23] 提供了一种表达能力更强的替代方案，可以解决这一问题。通过将控制与数据分离，TSL 使规约不仅能够表达事件的时间序关系，还能描述应用于任意类型变量的数据变换。

在本文中，我们展示了配备数据变换的时序逻辑能够作为一种具有表达能力的形式化语言，用于描述复杂规约。由于规范挖掘是在有限轨迹上进行的，我们引入了 TSL$_f$，即 TSL 的有限前缀解释。TSL$_f$ 原生支持描述变量随时间发生的函数式更新，从而提升了 LTL$_f$ 的表达能力。

我们的规范挖掘算法针对一组输入轨迹运行，其中的变量可以取任意类型的值，而不仅限于布尔值。要推断规约，我们必须先发现随时间改变系统变量的函数，以及这些变量之间的时序关系。我们使用语法引导综合（SyGuS） [1]，根据输入输出示例自动发现函数。为了确定一组能够描述变量如何演变的函数，我们的流程会为每个时间步的每个变量合成一个函数。接下来，我们反复剪枝这一集合，直到找到能够以最精简的方式描述整个轨迹中数据变换的函数。最后，我们利用学习得到的变换集合，将数据轨迹提升为函数应用轨迹。此类轨迹随后便可使用标准挖掘算法处理，例如布尔子集覆盖 [8]。

我们在 OpenAI-Gymnasium ToyText 套件 [11], 上评估我们的方法，该套件是一组环境，智能体必须在其中进行时序推理才能到达目标状态。仅使用正轨迹和负轨迹，且无需任何领域特定知识、动作函数或细粒度奖励信号，我们的方法即可提取出能够完整刻画环境目标的规约。我们通过综合反应式控制器并将其部署为游戏智能体来验证这些挖掘得到的规约，仅使用少量（≤ 20）示例，便在留出配置上取得了完美胜率。我们的方法显著优于符号基线方法与神经网络基线，所需示例数量减少了一个数量级，同时在泛化问题上表现出更好的性能。

总体而言，本工作能够挖掘出比以往方法更具表达能力的时序性质；更广泛地说，它迈出了通向纯符号强化学习（RL）范式的一步：智能体与环境交互，通过挖掘所生成的轨迹进行学习，并借助形式化规约不断完善自身。总之，我们的贡献如下：

– 我们提出一种自底向上的综合算法，通过贪心输入交换策略发现能够覆盖完整执行轨迹的函数集合。– 我们介绍一个基于函数和数据类型的规范挖掘框架，使用 TSL$_f$——时序流逻辑的有限迹语义。– 我们在标准强化学习基准上表明，基于挖掘出的规约，我们能够综合出样本效率高且具有良好泛化能力的反应式程序。

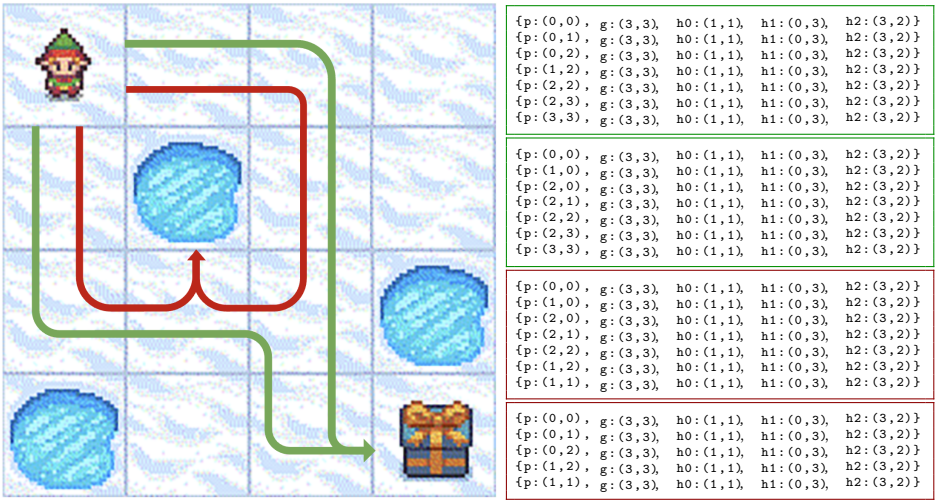

图 1： FrozenLake [11]。两条正轨迹（绿色）和两条负轨迹（红色）。

2 动机示例

现代对人工智能系统的评估强调分布外泛化，也就是将已学概念应用于训练期间未曾遇到的新情境的能力。ARC-AGI 基准已成为评估这一能力的重要测试平台，其任务基于离散网格，成功完成任务需要根据稀疏示例推断潜在规则 [15]。人类在这些任务上的准确率可达 80–90%，而最先进的大语言模型的表现则远远落后，准确率仅为 31% [24]。最近推出的 ARC-AGI3 将这一挑战扩展至“交互式推理基准” [32]：在这类网格游戏中，动作会随时间改变状态，能否成功取决于是否能够推断反应式规则。这类基准属于时序与关系推理问题。

我们以 FrozenLake为例说明我们的方法。这是 OpenAI-Gymnasium ToyText 套件中的一个典型网格世界任务 [11]，也属于这种时序与关系类别。如图 1 所示，智能体从起始格出发，必须避开表面上的洞并抵达目标。假设我们有两条正轨迹（成功抵达目标）和两条负轨迹（以掉入洞中告终）。每条轨迹由若干元组组成，记录五个变量（p、g、h0、h1、h2）的取值。其中，p 指玩家，g 指目标，h* 指洞。我们仅获得这些轨迹，没有任何特定领域的函数或谓词。

我们的流水线首先揭示博弈中存在的函数：玩家的坐标要么递增（+1），要么递减（-1）。随后，我们的挖掘算法同时推导出活性规约与安全规约：前者刻画最终必须发生的事情，后者陈述必须始终成立的事情。我们恢复了活性公式，它表明智能体最终必须到达目标。然而，根据这四条轨迹，所挖掘的安全规约指出

智能体不得向上移动。（注：`p[2]` 是 `p` 的第二个元组值。）

$$\varphi_{\text{live}} = \mathbf{F}(\texttt{(=) p g}),\quad \varphi^{(1)}_{\text{safe}} = \mathbf{G}\neg(\texttt{[p[2] <- (-1) p[2]]})$$

这个安全规约显然是虚假的，但它仍能区分给定样本中的正轨迹和负轨迹。如果我们根据$\varphi_{\text{live}} \wedge \varphi^{(1)}$ 安全的 合成一个反应式控制器并部署它，智能体确实会径直向下移动并掉进一个洞里。这次坠落产生了一条新的负轨迹。将其加入语料库并重新挖掘后，我们得到相同的活性公式，但安全规约经过了细化：

$$\varphi_{\text{live}} = \mathbf{F}(\texttt{(=) p g}),\ \varphi^{(2)}_{\text{safe}} = \mathbf{G}\neg(\texttt{(=) p h0} \vee \texttt{(=) p h1})$$

这个规约体现了该约束的本质（避开洞），但只识别出在洞**0**和洞**1**处发生的失败。根据这个规约合成控制器后，智能体会避开这两个洞，却掉进从未到访过的洞**2**。加入这个负样本后，我们提取出一个正确的规约

$$\varphi_{\text{live}} = \mathbf{F}(\texttt{(=) p g}),\quad \varphi^{(3)}_{\text{safe}} = \mathbf{G}\neg(\texttt{(=) p h0} \vee \texttt{(=) p h1} \vee \texttt{(=) p h2})$$

现在，合成控制器能够始终获胜。重要的是，由于规约采用关系式表达（将玩家坐标与洞的坐标进行比较，而不是记忆具体位置），它可以泛化到具有不同洞位布局和不同网格大小的棋盘，而这些棋盘在挖掘过程中从未出现过。

本质上，这展示了一种符号强化学习。我们并非通过梯度下降拟合策略，而是通过对轨迹进行规范挖掘来合成形式化世界模型。其结果是一个智能体，其行为受到所发现的时态与关系不变式的约束。

3 形式体系

我们介绍 TSL$_f$，即 TSL 的有限变体。为了学习关于数据变换的时态规约，我们需要一种原生支持时态算子和理论推理的逻辑。近年来，人们对这类逻辑的兴趣日益浓厚，例如 LTL$_f$-MT [42, 43, 41] 或 Reactive Program LTL [29, 28]。我们尤其关注 TSL$_f$，因为从工具支持的角度来看，它非常适用于构造反应式程序。

3.1 语法

在行为的有限前缀解释下，TSL$_f$ 的算子与其无限情形下的对应算子并无不同；不过，它们的语义有所不同。

TSL$_f$ 由一个签名 $\Sigma = (S,\ F,\ P,\ \Delta)$ 参数化，该签名由以下部分组成：

– 一个有限的集合 S，包含具有类型的流变量，它们表示数据演化。– 一个有限的集合 F，包含函数符号，每个符号都有类型签名 $\tau_1 \times \cdots \times \tau_n \to \tau$。– 一个有限的集合 P，包含谓词符号，其类型签名为 $\tau_1 \times \cdots \times \tau_n \to \text{Bool}$。– 一个背景理论 Δ，函数和谓词在该理论下进行解释。

我们采用 [14] 中的 TSL-MT 框架来解释和应用理论。在实验中，我们将 Δ 视为线性整数算术（LIA）的理论，但其他理论（例如数组理论）也同样适用。函数符号和谓词符号被解释为该理论上的具体实现，并由我们的挖掘过程发现。

在 TSL$_f$ 中，我们引入一个特殊的原子命题 END $\in \mathcal{P}$，它仅在轨迹的最后一个位置为真。这与在将 LTL$_f$ 转换为无限轨迹上的标准 LTL 时所用的 Alive 谓词互为对偶 [19]。

项。 我们在签名 Σ上定义两类项：

– 更新项（$\mathcal{T}_u$） $[s \leftarrow f(s_1, \ldots, s_n)]$，其中 s、s_1、…、$s_n \in S$，且 $f \in F$ 是一个函数，其类型签名与 s_1、…、s_n 的类型相匹配，并返回与 s 类型相同的值。$[s \leftarrow f(s_1, \ldots, s_n)]$ 表示在下一个时间步，通过将函数 f 应用于 s_1、…、s_n 的当前值来更新 s。– 谓词项（$\mathcal{T}_p$） $p(s_1, \ldots, s_n)$，其中 $p \in P$，且 s_1、…、$s_n \in S$ 是类型与 p 的签名相匹配的流。它表示一个谓词在当前时间步对其各项成立。谓词项包括 END。

公式。 TSL$_f$ 公式由以下语法定义：

$$\varphi ::= \tau \in \mathcal{T}_u \cup \mathcal{T}_p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \tag{1}$$

其中，$\mathbf{X}$（下一时刻）和$\mathbf{U}$ （直到）是标准的 LTL$_f$ 算子。我们列出常用的标准缩写：

– Boolean connectives: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ and $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$
– Temporal operators: $\mathbf{F}\varphi \equiv \top \mathbf{U} \varphi$ (eventually) and $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$ (always)

3.2 语义

我们在有限轨迹上解释 TSL$_f$ 公式。轨迹 是一个有限序列 $\sigma = \sigma_0\sigma_1 \cdots \sigma_n$，其中每个 $\sigma_i \subseteq \mathcal{T}_u \cup \mathcal{T}_p$都是在位置 i成立的项的集合。

良构轨迹。 若轨迹 σ 满足以下约束，则称其相对于 TSL$_f$中的更新是良构的：

1. 对于每个位置 $i < |\sigma| - 1$ （即每个非末尾位置），每个流变量都恰好更新一次。这是由更新之间的互斥性所保证的；直观而言，这表示同一个变量不能在同一个时间步 [22]被设置为两个不同的项：

$$\forall i \in [0, |\sigma| - 1)\ \forall s \in S\ \exists!\ u_s \in \mathcal{T}_u.\ u_s \in \sigma_i$$

2. 在最终位置 $|\sigma| - 1$，没有更新成立。直观地说，无法明确确定流变量在下一时间步的值，因此该值未定义：

$$\forall u \in \mathcal{T}_u.\ u \notin \sigma_{|\sigma|-1}$$

3. 特殊谓词 END 仅在最后一个位置成立。

$$\texttt{END} \in \sigma_i \iff i = |\sigma| - 1$$

满足关系。 满足关系 $\sigma, i \models \varphi$ （“公式 φ 在轨迹 σ的位置i 成立”）是递归定义的。轨迹 σ 满足 公式 φ，当且仅当 $\sigma, 0 \models \varphi$:

$\sigma, i \models [s \leftarrow f(s_1, \ldots, s_n)] \iff [s \leftarrow f(s_1, \ldots, s_n)] \in \sigma_i \sigma, i \models p(s_1, \ldots, s_n)$ $\iff p(s_1, \ldots, s_n) \in \sigma_i \sigma, i \models \mathit{END} \iff i = |\sigma| - 1 \sigma, i \models \neg\varphi \iff \sigma, i \not\models \varphi \sigma,$ $i \models \varphi_1 \wedge \varphi_2 \iff \sigma, i \models \varphi_1$和 $\sigma, i \models \varphi_2 \sigma, i \models \mathbf{X}\varphi \iff i + 1 < |\sigma|$和 $\sigma,$ $i + 1 \models \varphi \sigma, i \models \varphi_1 \mathbf{U} \varphi_2 \iff \exists j \geq i. \big(j < |\sigma| \wedge \sigma, j \models \varphi_2 \wedge \forall k. ($ $i \leq k < j \rightarrow \sigma, k \models \varphi_1)\big)$

3.3 可实现性

在反应式综合的背景下，如果存在根据输入/输出历史计算输出更新的函数，使得 φ 对所有可能的输入行为都满足，则规约 φ 可实现。对于 TSL，这通过将流划分为输入流 $I \subseteq S$ （由环境控制）和输出流 $O = S \setminus I$ （由系统控制） [23]来体现。对于 TSL$_f$，可实现性问题的定义类似，但针对的是有限时域策略。系统必须综合出更新函数，以保证对于任意有限输入都满足 φ。

TSL$_f$的可实现性问题与其无限对应问题类似，也是不可判定的。在 [23],中，将 Post 对应问题（PCP）归约为一个 TSL 规约，从而必然构造出一个不可判定的可实现性实例。该证明在 TSL$_f$ 语义下仍然成立，因为 TSL$_f$ 是在有限但任意的前缀解释下定义的。采用明确最大轨迹长度的严格有界表述是可判定的，但这超出了本文的范围。

尽管 TSL$_f$ 继承了 TSL 的语法，有限轨迹解释仍对可实现性施加了根本性限制。具体而言，就可实现规约类而言，TSL$_f$ 的表达能力严格弱于 TSL。

定理 1. *TSL$_f$ 的表达能力严格弱于 TSL*

这一差距同样存在于 LTL$_f$ 与 LTL [19]之间。该区别源于时态算子在有限轨迹上的语义与更新互斥约束之间的相互作用。考虑以下 TSL 规范:

$$\mathbf{G}\ (\ \mathbf{F}\ [x \leftarrow y]\ \wedge\ \mathbf{F}\ [x \leftarrow z]\ ).$$

在标准 TSL 语义下，这可实现为一个在 $[x \leftarrow y]$ 与 $[x \leftarrow z]$ 之间无限交替的控制器。然而，在有限前缀解释下，$\mathbf{G}(\mathbf{F}[x \leftarrow y])$ 要求在最后一个时间步 $[x \leftarrow y]$ 为真，因为在有限轨迹的每一步，最终都必须存在一个更晚的步骤使该更新成立。同样，$\mathbf{G}(\mathbf{F}[x \leftarrow z])$ 要求 $[x \leftarrow z]$ 在最后一个时间步为真。这违反了良构语义，具体而言是更新互斥，即在给定时间步，一个流变量只能被更新一次。形式证明见第 A 节。

```
{x: 0, y: 0, gX: 3, gY: 3, h0X: 1, h0Y: 1, h1X: 3, h1Y: 1, h2X: 3, h2Y: 2}
{x: 0, y: 1, gX: 3, gY: 3, h0X: 1, h0Y: 1, h1X: 3, h1Y: 1, h2X: 3, h2Y: 2}
{x: 0, y: 2, gX: 3, gY: 3, h0X: 1, h0Y: 1, h1X: 3, h1Y: 1, h2X: 3, h2Y: 2}
{x: 1, y: 2, gX: 3, gY: 3, h0X: 1, h0Y: 1, h1X: 3, h1Y: 1, h2X: 3, h2Y: 2}
{x: 2, y: 2, gX: 3, gY: 3, h0X: 1, h0Y: 1, h1X: 3, h1Y: 1, h2X: 3, h2Y: 2}
{x: 2, y: 3, gX: 3, gY: 3, h0X: 1, h0Y: 1, h1X: 3, h1Y: 1, h2X: 3, h2Y: 2}
{x: 3, y: 3, gX: 3, gY: 3, h0X: 1, h0Y: 1, h1X: 3, h1Y: 1, h2X: 3, h2Y: 2}
```

图2：OpenAI-Gymnasium FrozenLake 博弈轨迹。每行代表一个时间步，并编码环境状态。x 和 y 表示玩家坐标，gX 和 gY 表示目标坐标，h*X 和 h*Y 表示障碍物坐标。

4 挖掘过程

规范挖掘是从示范中推断规约 ϕ 的任务，使得正轨迹满足 ϕ 而负轨迹违反它。LTL$_f$规范挖掘作用于形如 $\alpha_1 \wedge \alpha_2$; $\alpha_2 \wedge \alpha_3$; $\alpha_1 \ldots$, 的布尔轨迹集合，其中 α_i 表示原子命题，分号分隔时间步。这在实践中存在局限，因为真实系统涉及定义在非布尔数据类型上的复杂函数、谓词和时序交互。例如，考虑来自 OpenAI-Gymnasium ToyTextFrozenLake 博弈的执行日志（图 2）。在此，甚至在尝试挖掘系统时序行为之前，我们需要确定数据值如何随时间更新。具体而言，我们首先需要假定哪些函数以及哪些输入被应用于不同变量，然后才能提取这些行为的时序依赖。

4.1 函数发现

给定一组轨迹，函数发现问题旨在找到一组函数F，用于解释整个集合中的变量演化。这是富数据规约挖掘的前提：如果不知道哪些函数能够解释数据变换，我们就无法构造定义变量的更新项。对于除最后一个时间步之外的每个时间步，我们的环境中的每个变量 $v \in S$ 都有一个更新项 $[v \leftarrow f\boldsymbol{w}], \quad f \in F,$ $\boldsymbol{w} \in S^{|\boldsymbol{w}|}$，且每个非终止时间步均如此。

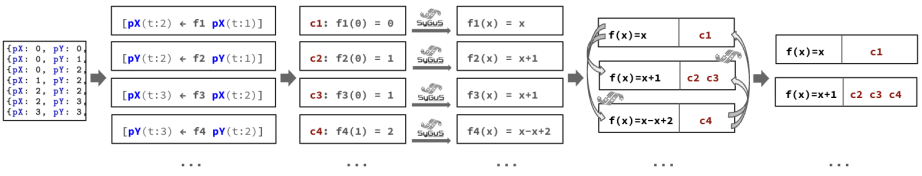

图3：函数发现过程。选定输入输出对后，Sy-GuS 求解器会针对每个约束确定唯一函数。根据合成函数合并约束；此时，较小的约束集会尝试并入较大的约束集，从而消除虚假函数。

算法 1：综合覆盖函数

输入：日志 Π，变量 S, $k_{\max}$ （最大元数）
输出：函数集 F，成功标志 s
对每个 $\pi \in \Pi$ **执行**
  // 阶段 1：为 π 提取状态转移
  $T_\pi \leftarrow S \times [1, |\pi|)$;
  对每个 $(v,t) \in T_\pi$ **执行**
    $\text{alts}_\pi[(v,t)] \leftarrow \{(\text{val}^\pi_{t-1}(\boldsymbol{w}), \text{val}^\pi_t(v)) \mid \boldsymbol{w} \in S^\ell, \ell \in [1, k_{\max}]\}$;
    $\text{rec}_\pi[(v,t)] \leftarrow$ 备选项$_\pi[(v,t)][0]$;
  // 阶段 2：单例综合
  对每个 $(v,t) \in T_\pi$ **执行**
    $f_\pi[(v,t)] \leftarrow \text{SyGuS}(\text{rec}_\pi[(v,t)])$;
  // 阶段 3：分组与合并
  $G_\pi = \{\{(v,t) \in T_\pi \mid f_\pi[(v,t)] = f\} \mid f \in$ 图像$(f_\pi)\}$;
  $F_\pi \leftarrow$ 自底向上合并$(G_\pi, \text{rec}_\pi, \text{alt}_\pi, m_{\max})$;
return $\bigcup_{\pi \in \Pi} F_\pi$

我们提出了一种自底向上的贪心函数发现过程。我们从轨迹的单元素划分开始，将每个变量转移视为一个拥有自身函数的约束。随后，我们迭代地合并能够由同一个函数统一的组；若合并失败，则探索备选函数输入。算法在无法进一步成功合并时终止，得到一个覆盖所有变量在所有时间步取值的完整函数集（相对于理论语法而言），且该函数集在局部上是最小的。这种贪心策略避免了穷举，同时仍能发现紧凑的函数集。

我们算法所依赖的核心原语是语法引导综合 （SyGuS） [1]。SyGuS 求解器（我们使用 CVC5 [6]）以一组约束$\{f(x_1) = y_1, \ldots, f(x_k) = y_k\}$以及定义候选函数空间的语法作为输入，并构造一个满足所有约束的函数 f 。例如，给定示例 $\{(0,1), (1,2), (2,3)\}$，CVC5 综合出 $f(x) = x + 1$。如果 SyGuS 求解器未能找到满足输入输出约束的函数，我们便假定不存在这样的函数，因而必须选择另一组能够覆盖该轨迹的约束。根据经验，我们发现 0.1 秒是足够的超时时间。

算法 1 展示了我们的自底向上综合过程。该算法以一组轨迹、最大函数元数和最大函数数量为输入，并输出一组发现的函数。该算法分三个阶段进行：（1）初始化候选变量转移并生成输入分组；（2）通过单例综合，寻找能够解释每个选定转移的独立函数；（3）对转移约束集进行分组。在第一阶段，算法针对每个时间步，为给定变量确定可能的输入（前一时间步的变量）。选定一种配置后，我们便针对每个转移独立地综合出唯一的函数。

算法 2 实现了合并逻辑。从单元素划分开始，算法会贪心地尝试将较小的组与较大的组合并

算法 2： 自底向上合并

输入： 组 G，记录 rec，备选项 alts
输出： 函数集 F
$G \leftarrow$ 排序(G, $\lambda x.y.|x| < |y|$)；　　// 将 G 组按从小到大的顺序排序
重复
　$merged \leftarrow$false;
　对于每个 $g_i \in G$ **执行**
　　// 尝试与当前输入赋值合并
　　对于每个 $g_j \in G$ *where* $|g_j| > |g_i|$ **执行**
　　$R \leftarrow \{\text{rec}[(v,\ t)] \mid (v,\ t) \in g_i \cup g_j\}$;
　　如果 $\mathit{SyGuS}\ (R) \neq \mathit{null}$ **则**
　　$g_j \leftarrow g_j \cup g_i$;　$g_i \leftarrow \emptyset$;
　　$merged \leftarrow$真；**中断**;
　　// 如果合并失败，请尝试其他输入源
　　如果 $g_i \neq \emptyset$ **则**
　　对 $(v,\ t) \in g_i$ **执行**
　　对每个 $r \in \mathit{alts}[(v,t)]$ **执行**
　　rec$[(v,\ t)] \leftarrow r$；　　// 切换到替代方案
　　对每个 $g_j \in G$ *where* $|g_j| > |g_i|$ **执行**
　　$R \leftarrow \{\text{rec}[(v,\ t)] \mid (v,\ t) \in g_i \cup g_j\}$;
　　如果 $\mathit{SyGuS}(R) \neq \mathit{null}$ **则**
　　$g_j \leftarrow g_j \cup g_i$;　$g_i \leftarrow \emptyset$;
　　$merged \leftarrow$true；**中断**；；　　// 中断以重复
直到 $\neg merged$;
返回 $\{f_i \mid g_i \in G, g_i \neq \emptyset\}$;

，方法是检查是否存在一个函数能够同时解释两组约束。例如，对于给定的约束 f(1)=2，CVC5 有时会发现一个伪函数，例如 $f(x) = (x - x + 2)$。将这个输入输出对与其他 (+1) 对结合起来，可以排除这种伪情况。我们优先合并较小的组，因为它们更常对应伪关系。当使用当前输入源分配无法直接合并时，算法会为较小组中的状态转移探索其他输入变量，并假定该变量是由于某个其他输入而更新的。这相当于尝试不同的假设，判断在该时间步哪些变量是输出变量的输入。此过程会持续进行，直到当前这组组配置无法再找到更多合并为止，最终得到一个局部最小函数集。

该算法保证能够找到一个函数集，解释所有观测到的状态转移。在最坏情况下，如果没有任何合并成功，算法就会为每个状态转移返回一个函数（即单元素划分）。贪心合并策略无法保证全局最小性；不同的输入源赋值或合并顺序可能会产生更少的函数，但算法 2 的自底向上合并过程能够高效地趋向紧凑的函数集。

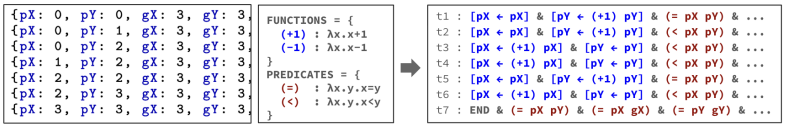

图 4：构建格式良好的 TSL$_f$ 轨迹。在学习了系统的函数后，我们对系统行为的有效解释进行建模。为了使轨迹格式良好，每个变量只能有一次更新（直到最后一个时间步）。

算法 3： ConstructTraces

输入： 日志 Π、变量 S、函数 F、谓词 P、排名 rank
输出： TSL$_f$ 轨迹 $\{\sigma_1, \ldots, \sigma_m\}$
result $\leftarrow$ [ ];
对于每个 $\pi \in \Pi$ **执行**
  $\sigma \leftarrow$ [ ];
  对于每个 $i \in [0, |\pi| - 1)$ **执行**
    $\sigma_i \leftarrow \{\}$;
    对于每个 $v \in S$ **执行**
      对于每个 $(f,\ \boldsymbol{w}) \in rank[v]$ **执行**
        如果 $f(val_i^{\pi}(\boldsymbol{w})) = val_{i+1}^{\pi}(v)$ **那么**
          $\sigma_i \leftarrow \sigma_i \cup \{[\mathrm{v} \leftarrow \mathrm{f(w)}]\}$; **中断;**
    对每个 $p \in P$ **执行**
      如果 $p(val_i^{\pi}(\boldsymbol{w}))$ **那么** $\sigma_i \leftarrow \sigma_i \cup \{\mathtt{p(w)}\}$;
  $\sigma_{|\pi|-1} \leftarrow \{\mathrm{END}\}$; // 仅在最后一个时间步使用谓词和 END
  对每个 $p \in P$ **执行**
    如果 $p(val_{|\pi|-1}^{\pi}(\boldsymbol{w}))$ **，则** $\sigma_{|\pi|-1} \leftarrow \sigma_{|\pi|-1} \cup \{\mathtt{p(w)}\}$;
  结果 $\leftarrow \sigma ::$ result;
返回 *result*

4.2 提升良构轨迹

发现函数集 F 后，我们现在构造格式良好的 TSL$_f$ 轨迹。这需要将原始日志序列转换为轨迹 $\sigma = \sigma_0\sigma_1 \cdots \sigma_n$，其中每个 $\sigma_i \subseteq \mathcal{T}_u \cup \mathcal{T}_p$ 都包含位置 i处的更新和谓词项。请回想，要使轨迹格式良好，我们必须确定变量如何演化：每个时间步 $i < |\sigma| - 1$ 中，每个变量都只能有一个更新。当存在多个有效更新时，就会出现一个难题。请看图 2 中时间=4处的 y 。假设 $(+1) \in F$, $[y \leftarrow y]_{t_4}$是一个有效更新，$[y \leftarrow x+1]_{t_4}$也同样是有效更新。虽然我们知道前者才是“正确赋值”，但这并未明确体现出来。

我们通过采用一种选择策略来解决这个问题：确定单一解释，并根据函数应用的频率进行排序。对于每个变量 v，我们根据函数-输入对在所有轨迹中作为 v 的有效更新出现的频率进行排序。如果在某个时间步有多个有效更新，我们就选择排名最高的选项。该分数根据所有给定日志汇总。算法 3 对这一过程进行了形式化描述，算法 4 则介绍了排序机制。

算法 4： 计算排名

输入： 日志 Π、变量 S、函数 F
输出： 排名 rank
对于每个 $v \in S$ **执行**
    rank$[v] \leftarrow \{\}$;
    对每个 $\pi \in \Pi$ **执行**
        对每个 $t \in [0, |\pi|)$ **执行**
            对每个 $(f, \boldsymbol{w}) \in F \times InputCombinations(v)$ **执行**
                如果 $f(val_t^\pi(\boldsymbol{w})) = val_{t+1}^\pi(v)$ **则**
                    rank$[v][(f, \boldsymbol{w})]$ ++;
    排序(排名$[v], \lambda x.y.x > y$) ; // 排序依据[v] 降序
返回 *rank*

由于候选配置有无限多个，因此需要发现函数；而谓词则与数据类型相关。例如，整数定义了相等关系（=）和序关系（<）。构造良构轨迹时，我们会在每个时间步对所有同类型变量应用这些谓词。

4.3 学习规约

有了格式良好的 TSL$_f$ 轨迹，我们现在可以对系统行为进行布尔化解释。这使我们能够调用 Bolt 等规约挖掘求解器 [8]，将更新和谓词项视为原子命题。Bolt 是一种先进的挖掘工具，它执行两阶段有界搜索：首先枚举大小不超过给定上限的时序公式，然后将布尔组合评估为布尔子集覆盖问题的一个实例。

要找到判别性规约，这已经足够。然而，要合成一个反应式 程序，最小的判别性公式往往规约不足。例如，在 FrozenLake 上，我们通常只会挖掘目标条件（**F (=) p g**），因为这是最小的判别性公式。然而，根据这一规约合成控制器会得到一个规约不足的智能体：该公式对反应条件（即始终避开洞）只字未提。

我们需要的是一种规约，既能描述智能体必须实现什么目标，也能描述它必须如何反应。这促使我们将挖掘问题分解为两个部分：一个活性 条件$\mathbf{F}\psi$，用于编码最终要达到的目标；以及一个安全性 条件$\mathbf{G}\,\varphi$，用于编码始终成立的不变式。二者共同构成$\mathbf{G}\varphi \wedge \mathbf{F}\psi$ 规约，可据此合成获胜控制器。这种结构与秩 1 广义反应性规约 [9]相契合。

我们扩展 Bolt，以挖掘安全与活性规约。对于活性规约，我们执行相同的有界搜索，但将候选公式 ψ 评估为 $\mathbf{F}\psi$，以寻找最小的判别性公式。对于安全性挖掘，我们将候选公式 φ 包装为 $\mathbf{G}\varphi$ 进行评估，并进一步将搜索语法限制为布尔联结词（$\wedge$、 $\vee$、 $\neg$、 $\rightarrow$、 $\leftrightarrow$）和下一时刻（$\mathbf{X}$），以确保得到的性质完全是不变性质。找到每种类型中最小的公式后，搜索即终止。

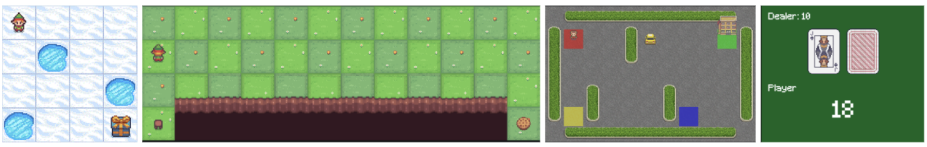

图 5：完整的 OpenAI-Gym ToyText [11] 套件。我们在广义实例上进行评估。从左到右：FrozenLake、CliffWalking、Taxi和 Blackjack。

5 评估

我们在 OpenAI-Gym ToyText 基准套件中的所有环境上，将 TSL$_f$ 规约挖掘作为一种符号策略学习框架进行评估 [11]。我们从轨迹中挖掘规约，合成反应式控制器，并将其部署到未见过的环境配置中。轨迹示例和生成流程见 B 节。我们汇总了满足动作机制（合法移动）且实现目标的实例总数。这定量评估了我们的方法发现这些环境的动作函数和目标机制的能力。

基线方法 我们以被动式模仿学习方法作为基准。这些基线方法映射到相同的任务，但其学习范式与我们不同，因为它们从状态-动作样本中学习局部策略。我们还注意到，这些方法仅从正示范中学习，而我们的方法则从正轨迹和负轨迹中学习具有判别性的规约。

– **随机 Mealy 机学习（Alergia）：** Alergia [12] 是一种经典算法，通过合并前缀树接受器中统计上兼容的状态，从正例中学习概率自动机。由于我们的演示可能会在相同状态下呈现不同动作（存在多条通往目标的有效路径），因此需要进行随机自动机学习。我们使用 AALp Alergia 实现 [35]，并根据转移概率采样动作；对于未见过的状态，则均匀随机地选择动作。– **行为克隆（神经网络）：** 行为克隆代表了模仿学习的神经网络方法 [39, 31]，其应用包括游戏对战 [30]和机器人学 [44]。我们构建了一个包含两个隐藏层（每层 64 个单元，使用 ReLU 激活函数）的前馈神经网络，并通过对状态-动作对进行监督学习来训练。训练使用交叉熵损失、Adam 优化器（$\alpha = 0.001$），并基于 20% 的验证集划分进行早停。– **决策树（CART）：** 一个在状态-动作对上训练的 CART 决策树 [10] 。我们采用基尼不纯度划分，并使用代价复杂度剪枝。– **位爆破（LTL-BB）：** 为单独考察 TSL$_f$ 作为一种具有表达能力的抽象机制的影响，我们将其与位爆破 LTL$_f$ 挖掘进行比较：将整数变量编码为布尔位，并将其视为原子命题。

我们的方法与这些基线方法的一个区别在于，TSL$_f$ 挖掘能够发现动作语义，而这些基线方法假设动作标签已给定。通过将任务表述为“给定一个状态，从已知动作中选择”，这些基线方法获得了额外的监督，而我们的方法必须从原始轨迹中推断这些信息。

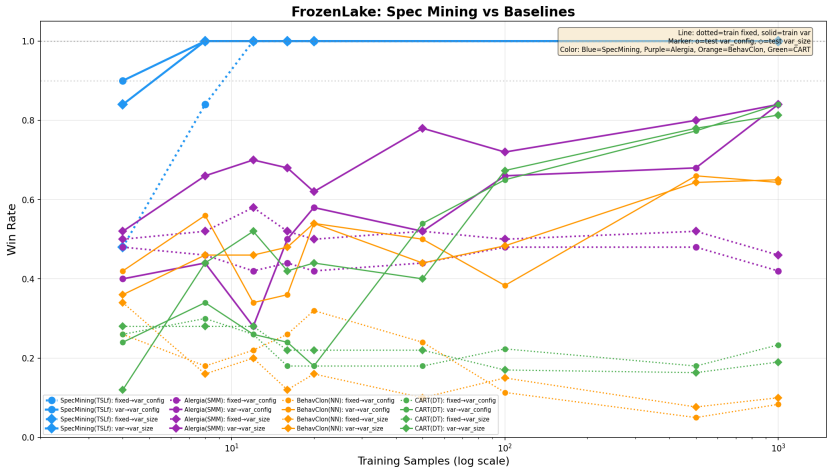

图6： FrozenLake 不同配置结果。规范挖掘比模仿学习基线方法更具样本效率，且泛化能力更好。

策略综合与部署 给定挖掘出的 TSLf 规约 φ_n，我们使用 Issy [28]综合一个反应式控制器。Issy 将 φ_n 综合为有限状态转换器，将环境观测映射为动作。我们在未见过的测试配置上部署这一学习得到的策略。对于网格世界游戏，我们将发现的更新项和挖掘出的活性-安全规范嵌入一个精简的骨架模板中，该模板规定了起始坐标和网格边界。二十一点也采用类似的模板。模板见 C 节。

结果 我们报告了 ToyText 套件上的结果，评估了对未见配置的泛化、样本效率，以及时序关系抽象相对于局部状态-动作学习的性能。对于每个环境，我们与学习基线方法进行比较，并检查 LTL-BB 生成的规约。学习与综合时间的统计数据见 E 节

在 FrozenLake 中，玩家在网格上从起点导航到目标，且不踩到洞。固定实例是一个 4×4 网格，洞位于 (1,1)、(0, 3) 和 (3, 2)，目标位于 (3,3)。我们评估对未见棋盘的泛化：变量置信度在网格上随机化目标/洞的位置，可变规模还使维度从 3×3 变化到 5×5。TSLf 从 n 条正轨迹和 n 条负轨迹（共计 $2n$）中挖掘，而模仿基线从 n条正轨迹中学习。

表 1 报告了两种泛化设置下 50 个测试实例的胜率。在所有配置下，TSL 规范挖掘f 仅用 24 条轨迹（12 条正向、12 条负向）就达到了 100% 的准确率，完美学会了既能到达目标状态又能避开洞的环境策略。在变化配置上训练的基线方法使用 1000 个样本后，准确率停留在 70–85%；而在固定配置上训练时，没有任何方法的准确率超过 50%。

差距反映了每种方法所学习的内容。模仿基线将状态映射到动作：在洞位于 (1, 1)，(0，3)，(3，2) 的情况下训练出的策略会记住这些避让动作，但洞的位置变化时并不具备迁移能力。

方法	训练	Test	4	8	12	16	20	50	100	500	1000
SpecMining(TSL$_f$)	固定变量_置信度	变量_置信度								-	-	-	-
	变量_置信度	变量_置信度								-	-	-	-
	固定变量_大小	变量_大小 2									-	-	-
		变量 大小 42								-	-	-	-
Alergia（SMM）	固定变量_置信度	变量_置信度									24	24	21
	变量_置信度	变量_置信度										34	42
	固定变量_大小	变量_大小 2									25	26	23
		变量 大小 26									36	40	42
行为克隆（神经网络）	固定变量_置信度	变量_置信度			11						6	3	4
	变量_置信度	变量_置信度									19	33	32
	固定可变规模	可变规模 1			10	6	8	5			8	4	5
		可变规模 18									24	32	32
CART（决策树）	固定变量_置信度	变量_置信度						9	9	9	11	9	12
	变量_置信度	变量_置信度										39	42
	固定可变规模	可变规模	14								8	8	9
		可变规模	6	22	26	21	22	20			34	39	41

表 1： FrozenLake 胜率（50 个测试实例中的结果）。可变置信度：_固定的 4×4棋盘，洞穴/目标位置各异。可变规模： 3×3 到 5×5 个棋盘，位置各异。各列显示训练样本数量。

方法活性		安全性
TSL$_f$	F((eq x goalx) ∧ (eq y goaly))	G ¬((eq x hole1x) ∧ (eq y hole1y)) G(((eq x hole0x) ∧ (eq y hole0y)) ↔ (((eq x hole1x) ∧ (eq y hole1y)))) G(((eq x hole2x) ∧ (eq y hole2y)) ↔ (((eq x hole0x) ∧ (eq y hole0y)) ∨ ((eq x hole1x) ∧ (eq y hole1y))))
LTL-BB	F(x b1) _ F((y b0) ∧ (X x b1)) F(G ((y b0) ∧ (y b1))) F((goalY b0) ↔ (hole1Y b1))	G ((X goalX b2) → (x h1)) G ((y b0) → ((y b1) ↔ (X y b0))) G ((X goalX b2) → ((y b0) ∧ (y b1))) G((goalY b0) ↔ (hole1Y b1))

表2：TSL$_f$ 和 LTL-BB 在 FrozenLake 上发现的独特规约。

TSL$_f$ 会挖掘关于谓词的关系规约，这些谓词用于比较玩家和洞的坐标。表2展示了挖掘出的安全公式（双条件式编码了互斥；只有玩家不经过任何一个洞时，该公式才成立）。由于规约是基于关系定义的，因此它可以迁移到新的洞位布局。位爆破会产生诸如**F**(x_{b1})（“最终 x 的第 1 位为真”）这样的公式。这些公式在语法上是有效的 LTL$_f$，但失去了任何有用的语义解释。

在 CliffWalking 中，玩家从 (0,0) 出发，必须绕过悬崖前往最右侧的 (w,0) 点。固定实例规定了一个12×4网格，玩家位于 (0, 0)，目标位于 (11,0)，悬崖高度为 1。我们改变悬崖高度 $h\in[1,3]$、棋盘宽度 $w\in[3,12]$ 和棋盘长度 $l\in[4,6]$。

在变化配置上训练的 TSL$_f$ 实现了完美泛化。然而，在固定配置（悬崖高度为 1）上训练，在不同尺寸测试中仅达到 34%。从固定配置训练中挖掘出的规约-(表 4）表述为“ $x=0$ 或 $y=0$ 蕴含 $x=$ 目标x”。虽然这概括了固定情况下的获胜策略，但悬崖高度增加会使危险区域扩展至 $[0,h-1]$，从而改变变量与悬崖之间的语义关系。表4还表明，模仿学习基线方法在可变规模训练中学得了完美策略，

方法	训练	Test	4	8	12	16	20	50	100	200	500	1000
SpecMining(TSL^f)	固定	可变规模 1								-	-	-
	可变规模	可变规模 5								-	-	-
	变量移动 固定	可变规模 17	1							-	-	-
	变量移动 太小	可变规模 0								-	-	-
Alergia（SMM）	固定	可变规模 1								18	19	20	20
	可变规模	可变规模 5									**50**	**50**	**50**
	变量移动 固定	可变规模 22	2							14	3	0	0
	变量移动 太小	可变规模 2								18	4	0	0
行为克隆（神经网络）	固定	可变规模	5	5	2	7	7	7	7	8	8	7
	可变规模	可变规模 2								47	47	23	17
	变量移动 固定	可变规模 7		7	7	3	7	3	3	3	0	3
	变量移动 太小	可变规模 1								33	23	17
CART（决策树）	固定	可变规模	2	2	2	2	2	2	2	2	2	2
	可变规模	可变规模 2								37	23	20
	变量移动 固定	可变规模 3		3	3	0	0	3	3	3	3	3
	变量移动 太小	可变规模 1								37	23	20

表3： CliffWalking 在固定棋盘及不同棋盘配置上的训练获胜次数。在50个可变规模实例上测试（悬崖高度 $h \in [1,3]$、棋盘宽度 $w \in [3,12]$、棋盘长度 $l \in [4,6]$）。

方法 活性		安全性
$\mathbf{TSL}_f$	$\mathbf{F}$((eq x goalx) $\wedge$ (eq y goaly))	$\mathbf{G}$ ((eq x goaly) $\vee$ ((eq y goaly) $\rightarrow$ (eq x goalx)))
		$\mathbf{G}$((eq x goalx) $\vee$ ((lt y cliffHeight) $\rightarrow$ (eq x goalx)))
LTL-BB	$\mathbf{F}(\mathbf{G}$((x b0) $\wedge$ (x b1)))	$\mathbf{G}$((**X** cliffHeight b1) $\rightarrow$ ((x b0) $\wedge$ (x b1)))
	$\mathbf{F}(\mathbf{G}$((x b0) $\wedge$ (x b3)))	$\mathbf{G}$((**X** cliffHeight b1) $\rightarrow$ ((x b0) $\wedge$ (x b3)))
	$\mathbf{F}(\neg(($**X** x b3) $\rightarrow$ (y b1)))	$\mathbf{G}$ ((**X** cliffHeight b2) $\rightarrow$ (x b3))

表4：TSL^f 和 LTL-BB 在 CliffWalking 上发现的独特规约。

但在固定训练中却没有。CliffWalking 的位爆破规约也表现出与 FrozenLake类似的语义局限。

我们还额外评估了一个具有非标准动力学的修改后的环境变量移动。在此，向右移动会更新 $x \mapsto 2x+1$，向上移动会改变 $y \mapsto y+2$。函数发现恢复了这些变换，并构建了恰好包含更新 $[x \leftarrow x]$, $[x \leftarrow (x+x)+1]$, $[x \leftarrow x-1]$, $[y \leftarrow y]$,$[y \leftarrow y+2]$, $[y \leftarrow y-1]$ 的 TSL^f 轨迹。合成控制器会自动遵循这些改变后的动力学，而无需修改挖掘或合成流程。

值得注意的是，在此设定下，基线性能会随着额外样本的增加而下降。在使用移动函数 $x \mapsto 2x+1$ 和 $y \mapsto y+2$ 时，生成多样化轨迹需要大量回溯（每一次较大的向前步进都需要多次较小的向后步进，才能产生不同的轨迹）。在此，Alergia 从状态-动作频率中学习转移概率，会越来越多地观察到同一状态上的向后移动，从而学到这种状态-动作映射。决策树和神经网络表现出类似的性能退化。使用 TSL^f 进行规范挖掘更为稳健，因为它基于时序关系全局结构进行推理，而非局部转移频率。

在 Taxi 中，智能体必须在四个指定位置之一接载乘客，并将其送到带有墙壁的 5×5 网格中的目的地；如果把乘客送到错误的目的地，则会失败。固定案例中，智能体从 (2，2) 出发，乘客位于 (0，0)，目的地位于 (4，0)，而错误的

方法	训练	Test	4	8	12	16	20	50	100	200	500	1000
SpecMining(TSL$_f$)	固定	变量_pos 2_31	5								-	-	-	-
	变量_pos	变量_pos 2									-	-	-	-
Alergia（SMM）	固定	变量_位置									17	17	23	19
	变量_位置	变量_位置 2										28	30	31
行为克隆（神经网络）	固定	变量_pos (			0	0	1	0	0	2	1	0	0
	变量_pos	变量_pos 2			4	3	12	0	9	9	14	17	14
CART（决策树）	固定	变量_pos 1			1	1	2	1	1	1	1	1	1
	变量_pos	变量_pos 2			0	1	2	2	2	4	13	10	13
行为克隆（神经网络）*	固定	变量_pos 5			7	2	5	4	3	2	2	3	3
	变量_pos	变量_pos 3 1								20	17	28	24
CART（决策树）*	固定	变量_位置 1			2	2	1	3	1	4	4	0	0
	变量_位置	变量_位置 3					13	5	17	18	12	26	24

表 5：TAXI 在固定棋盘和多种棋盘配置上训练所得的胜利次数。在 50 个变量 位置实例上测试，改变墙壁、乘客和目的地的位置。$*$ 表示一个由两个基线方法串联而成的双阶段系统。

方法	活性	安全性
TSL$_f$	**F** (eq x DEST x $\wedge$ eq y DEST y) **F** ((eq x RED X $\wedge$ eq y RED_Y) $\wedge$ **F**(eq x DEST X $\wedge$ eq y DEST-Y))	**G** (eq x YELLOW x $\wedge$ eq y YELLOW y) **G**((eq x RED X $\wedge$ eq y RED Y) $\leftrightarrow$ (eq x YELLOW X_$\wedge$ eq y YELLOW Y))
LTL-BB	**F**((x b0) $\leftrightarrow$ (RED x_b1)) **F**((DEST x b1) $\leftrightarrow$ (((DEST y b0) $\leftrightarrow$ (RED_y_b1)) $\rightarrow$ (DEST-x-b0)))	**G**((x b0) $\leftrightarrow$ (RED x_b1)) **G**((DEST x b1) $\leftrightarrow$ (((DEST y b0) $\leftrightarrow$ (RED y b1)) $\rightarrow$ (DEST x b0)))

表 6：TSL$_f$ 和 LTL-BB 在 TAXI 上挖掘出的独特规约。LTL-BB 在 $n \geq 12$ 上挖掘超时。

下车点位于 (0，4) 和 (3，4)。表 5 报告了变量位置泛化下的结果，_其中起始位置、墙壁布局、乘客位置和目的地各不相同。

这个环境很有意思，因为其目标条件具有时间顺序：智能体必须先前往乘客所在处，然后前往目标位置。使用 1000 个样本时，无论是在固定配置还是变化配置上训练，模仿基线的表现都未超过 32%。这种失败源于表征能力：这些方法学习的是从观测到动作的无状态映射，无法将“先去乘客所在处，再去目标位置”编码为策略。为隔离这一影响，我们评估了两阶段基线（神经网络*、决策树*），它们将每个阶段的独立模型串联起来。一个模型在前往乘客所在处的轨迹上训练，另一个则在从乘客所在处前往目的地的轨迹上训练。表现提升至 56%，这表明它们难以捕捉时间结构。TSL$_f$ 仅用 24 条轨迹便直接挖掘出序列目标：最终到达乘客所在处，然后最终到达目的地（见表 6）。LTL-BB 在 $n \geq 12$ 条轨迹时超时，否则会推导出虚假的比特等价关系。

BLACKJACK是 Gymnasium 环境中的最后一个，与前面的网格世界有着根本性的不同。这里没有空间坐标，也没有需要发现的移动函数，游戏结果具有随机性，并且每个回合最多持续三步。它并不适合进行时序关系推理：决策取决于局部状态（当前手牌、庄家牌），没有可供利用的序列结构。尽管如此，TSL$_f$ 仍展现出了挖掘策略的能力。

方法	策略	4	8	12	16	20
SpecMining(TSL$_f$)	阈值	**50 (22) 50**				
	保守 **50 (24) 50 (**					
	基础	44 (22)	44 (22)	**50 (25)**	**50 (25)**	**50 (25)**
Alergia（SMM）	阈值	**50 (22) 50**				
	保守 **50 (24) 50 (**					
	基础	44 (22)	**50 (25)**	**50 (25)**	**50 (25)**	**50 (25)**
行为克隆（神经网络）	阈值	46 (18)	**50 (22)**	44 (21)	**50 (22)**	46 (18)
	保守 29(23)		42 (23)	47 (23)	46 (25)	45 (23)
	基础	44 (22)	34 (23)	49 (23)	45 (24)	44 (22)
CART（决策树）	阈值	46 (18)	**50 (22)**	**50 (22)**	**50 (22)**	**50 (22)**
	保守 29(23)		47 (23)	47 (23)	47 (23)	47 (23)
	基础	44 (22)	29 (24)	43 (22)	45 (23)	49 (23)

表7：在 50 个实例上，针对不同训练集规模的 BLACKJACK 策略遵循度（胜率）。随着策略变得更加复杂，胜率会提高。

方法策略		安全性规约
TSL$_f$	阈值	$\mathbf{G}\ \neg((\texttt{lt c sThresh}) \leftrightarrow (\mathbf{X}\ \texttt{stood}))$
	保守	$\mathbf{G}\ ((\texttt{lt c sThresh}) \rightarrow (\mathbf{X}((\texttt{lt c sThresh}) \leftrightarrow (((\texttt{gte dealer lo}) \wedge (\texttt{lte dealer med})) \vee (\texttt{lt c sWeak})))))$ $\mathbf{G}\ \neg((\mathbf{X}\ \texttt{stood}) \leftrightarrow ((\texttt{lt c sThresh}) \wedge (((\texttt{gte dealer lo}) \wedge (\texttt{lte dealer med})) \rightarrow (\texttt{lt c sWeak}))))$ $\mathbf{G}\ ((\texttt{lt c sWeak}) \vee (\mathbf{X}\ ((\texttt{eq c sWeak}) \leftrightarrow [\texttt{c} \leftarrow \texttt{c}])))$
	基础	$\mathbf{G}\ \neg((\mathbf{X}\ \texttt{stood}) \leftrightarrow ((\texttt{lt c sThresh}) \wedge (((\texttt{gte dealer lo}) \wedge (\texttt{lte dealer med})) \rightarrow (\texttt{lt c sWeak}))))$
LTL-BB	阈值	$\mathbf{G}((\texttt{c b4}) \rightarrow (\mathbf{X}\ \texttt{stood}))$
	保守	$\mathbf{G}((\texttt{dealer b2}) \leftrightarrow (\texttt{c b3}))$
	基础	$\mathbf{G}((\texttt{dealer b1}) \leftrightarrow (\mathbf{X}\ \texttt{dealer b1}))$

表 8：在 BLACKJACK 上由 TSL$_f$ 和LTL-BB 挖掘出的规约。**图例：** c=玩家牌总数，lo=2，med=6，sThresh=17，sWeak=12。

由于 BLACKJACK中的游戏结果具有非确定性，我们根据 策略遵循度来区分轨迹，按照游戏过程是否遵循三种复杂度递增的策略进行划分：阈值（如果手牌 ≥ 17则停牌）、保守（将庄家牌纳入决策）以及基础（完整的基础策略）。有关策略的形式化定义，请参见 D 节。由于不存在能够区分这些轨迹的确定性目标，因此没有可供挖掘的活性条件（活性挖掘会超时）。因此，我们只挖掘约束何时停牌的安全性规范。此外，我们注意到，TSL$_f$ 以关系方式在变量上挖掘谓词，但二十一点策略引用的是绝对阈值（例如 17）。为实现挖掘，我们向轨迹中添加常数（见表 8 图例）。提供这些常数后，挖掘器便能针对常数边界表达谓词。这反映了纯关系挖掘的局限性：TSL$_f$ 无法仅从轨迹中原生发现 17 是一个重要阈值。参数识别技术 [4] 能够从数据中推断谓词边界，为未来的研究提供了一个方向。

表 7 报告了 50 个测试实例上的策略遵循度。Alergia 达到了最优性能，这一结果符合预期，因为二十一点的决策是无记忆的，并且可以自然地表示为随机自动机映射。TSL$_f$ 的遵循度达到 100%，所需轨迹数为 8 至 24 条，具体取决于策略复杂度，并且成功学习了所有策略。位展开规范偶尔也能近似实现正确性：$\mathbf{G}(c_{b4} \rightarrow (\mathbf{X}$停牌$))$ 表示第 4 位（手牌 ≥ 16）与停牌相关。

6 相关工作

时序逻辑 线性时序逻辑 (LTL) [38] 及其有限轨迹变体 LTL_f [19]仍是时序规范的标准形式体系，为规划 [5, 2], 、流程挖掘 [33], 和运行时验证 [17]领域的研究奠定了基础。然而，这些逻辑是命题式的，将事件编码为布尔原子。时序流逻辑 (TSL) [23] 将控制与数据分离，从而支持针对未解释函数和谓词的规约；TSL-MT [21] 则将这些内容置于背景理论中。其他具备数据感知能力的扩展包括 LTL_f 模理论 [25], ，它支持单调计数器；以及 data-LTL_f [26], 一种由 SMT 支持、含有变量条件式的变体。信号时序逻辑 (STL) [7] 是另一种值得关注的范式，可对连续信号进行推理，并采用定量语义。我们的工作提出了 TSL$_f$，即 TSL 的一种有限轨迹解释。

规范挖掘规范挖掘从轨迹中提取时序性质。Texada [34] 执行基于模板的 LTL 挖掘；Bolt [8] 和 Scar- let [36] 通过集合覆盖形式扩展 LTL$_f$ 挖掘。这些方法基于布尔抽象，需要手工构造原子命题。在连续域中，STL 挖掘 [7] 也受到关注，参数识别技术 [4]通过信号学习阈值。

语法引导综合SyGuS [1] 技术根据语义约束和语法文法综合程序，从而支持 FlashFill [27]等示例编程技术。现代求解器（如 CVC5 [6]）支持线性算术和位向量等理论。SyGuS 已应用于运动规划 [13]。Das 等 [18] 将函数综合与自动机学习相结合，从网格世界观测中发现反应式程序。我们的工作以 SyGuS 作为函数发现的基础方法，随后将发现的函数提升为时序规范，从而融合语法引导综合与反应式综合。

7 结论

本文提出了一个框架，用于从执行轨迹中挖掘针对丰富数据类型的时序规范。我们利用语法引导综合来发现能够解释变量演化的函数，并将 TSL$_f$ 形式化为时序流逻辑的有限轨迹解释；由此，我们的方法能够挖掘同时捕捉数据变换和时序结构的规约。我们在 OpenAI-Gymnasium ToyText 基准上的评估表明，从挖掘出的规约综合得到的反应式控制器，在未见过的配置上能够达到完美准确率，同时所需样本数量显著少于被动学习基线。未来，一个自然的拓展方向是进一步形成闭环，实现完全符号化的强化学习：智能体主动生成轨迹，从经验中挖掘规约，并通过形式化综合迭代改进其行为。

参考文献

[1] Alur, R., Bodik, R., Juniwal, G., Martin, M.M., Raghothaman, M., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.：语法引导综合。载于：2013年计算机辅助设计中的形式化方法。第1–8页。IEEE（2013）[2] Aminof, B., De Giacomo, G., Sanchez, C., Rubin, S.：多智能体环境假设下的 LTL 综合。载于：第22届知识表示与推理原理国际会议（KR）论文集（2025），http://www.diag.uniroma1.it/~degiacom/papers/2025/KR2025adpr.pdf[3]Ammons, G., Bod´ık, R., Larus, J.R.：规范挖掘。《SIGPLAN 通讯》**37**(1)，4–16（2002年1月）。https://doi.org/10.1145/565816.503275, https://doi.org/10.1145/565816.503275[4] Asarin, E., Donz´e, A., Maler, O., Nickovic, D.：时序性质的参数辨识。载于：RV。第147–160页（2011）[5] Bacchus, F., Kabanza, F.：使用时序逻辑表达规划中的搜索控制知识。《人工智能》**116**(1-2)，123–191（2000）[6] Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., N¨otzli, A., 等：cvc5：一种通用且具备工业级性能的 SMT 求解器。载于：系统构造与分析工具和算法国际会议。第415–442页。施普林格（2022）[7] Bartocci, E., Mateis, C.,Nesterini,E., Nickovic, D.：信号时序逻辑规范挖掘综述。《信息与计算》**289**，104957（2022）[8] Bathie, G., Fijalkow, N., Matricon, T., Mouillon, B., Vandenhove, P.：LTL$_f$学习与布尔集合覆盖相遇（2025），https://arxiv.org/abs/2509. 24616[9] Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Saar, Y.：反应式(1)设计的综合。《计算机与系统科学杂志》**78**(3)，911–938（2012）。https://doi.org/https://doi.org/10.1016/j.jcss.2011. 08.007，纪念 Amir Pnueli[10] Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.：《分类与回归树》。Chapman and Hall/CRC（1984）[11]Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.：OpenAI Gym（2016），https://arxiv.org/abs/ 1606.01540[12] Carrasco, R.C., Oncina, J.：利用状态合并方法学习随机正规文法。载于：文法推断国际研讨会。第139–152页。施普林格（1994）[13] Chasins, S.E., Newcomb, J. L.：使用 SyGuS 综合反应式运动规划。载于：第5届综合研讨会（SYNT@CAV）论文集。EPTCS，第229卷，第3–20页（2016）

[14] Choi, W., Finkbeiner, B., Piskac, R., Santolucito, M.: 反应式综-合与语法引导综合能否成为朋友？载于：第43届ACM SIGPLAN编程语言设计与实现国际会议论文集。第229–243页。PLDI 2022，计算机协会，纽约州纽约市（2022）。https://doi.org/10.1145/ 3519939.3523429, https://doi.org/10.1145/3519939.3523429[15] Chollet, F.: 关于智能度量（2019），https://arxiv.org/ abs/1911.01547[16] Clarke, E.M., Emerson, E.A., Sistla, A.P.: 使用时序逻辑规约对有限状态并发系统进行自动验证。ACM编程语言与系统汇刊（TOPLAS）8(2)，244–263（1986）[17] Clarke, E.M., Grumberg, O., Peled, D.A.: 模型检测。麻省理工学院出版社，剑桥，马萨诸塞州（1999）[18] Das, R., Tenenbaum, J.B., Solar-Lezama, A., Tavares, Z.: 结合函数式综合与自动机综合以发现因果反应式程序。ACM编程语言论文集7(POPL)，1628–1658（2023）。https://doi.org/10.1145/3571249[19] De Giacomo, G., Vardi, M.Y.: 有限轨迹上的线性时序逻辑与线性动态逻辑。载于：第23届国际人工智能联合会议（IJCAI）论文集。第854–860页。AAAI出版社（2013）[20] Dossche, N., Coppens, B.: 利用结构相似性推断错误规约并检测漏洞。载于：第33届USENIX安全研讨会（USENIX Security 24）。第1885–1902页。USENIX协会，费城，宾夕法尼亚州（2024年8月）[21]Finkbeiner, B., Heim, P., Passing, N.: 模理论下的时序流逻辑。载于：软件科学与计算结构基础：第25届国际会议FOSSACS 2022，作为欧洲软件理论与实践联合会议ETAPS 2022的一部分召开，德国慕尼黑，2022年4月2日至7日，论文集。第325–346页。施普林格出版社，柏林，海德堡（2022）。https:// doi.org/10.1007/978-3-030-99253-8_17, https://doi.org/10.1007/ 978-3-030-99253-8_17[22] Finkbeiner, B., Klein, F., Pisk´aˇc, R., Santolucito, M.: 综合函数式反应式程序。载于：Haskell 2019论文集（2019），https: //arxiv.org/abs/1905.09825，展示了TSL综合中的控制流模型如何转化为函数式反应式编程（Applicative、Monad、Arrow框架）[23] Finkbeiner, B., Klein, F., Pisk´aˇc, R., Santolucito, M.: 时序流逻辑：超越布尔值的综合。载于：计算机辅助验证（CAV）2019，《计算机科学讲义》，第11561卷。第609–629页。施普林格（2019），https://doi.org/10.1007/978-3-030-25540-4_35，亦以CoRR abs/1712.00246（2017）形式提供[24] Foundation, A.P.: Arc-agi-2：面向前沿AI推理系统的新挑战。技术报告，ARC Prize基金会（2025），https: //arxiv.org/abs/2505.11831

[25] Geatti, L., Gianola, A., Gigante, N.: 有限轨迹上模理论的线性时序逻辑。载于：国际人工智能联合会议（IJCAI）论文集。第2641–2647页（2022），https: //www.ijcai.org/proceedings/2022/0366，介绍了LTL$_f$-MT，用一阶公式替换命题原子[26] Gianola, A., Montali, M., Winkler, S.M.: 面向数据感知流程挖掘的SMT技术。K¨人工智能（KI）（2025）。https://doi.org/10.1007/s13218-025-00890-z，综述了通过SMT/OMT和数据LTL进行的规范挖掘$_f$，用于数据感知流程分析[27]Gulwani, S.: 使用输入输出示例自动化电子表格中的字符串处理。载于：第38届ACM SIGPLAN-SIGACT编程语言原理年会（POPL）论文集。第317–330页。ACM（2011）。https://doi.org/10.1145/1926385.1926423[28] Heim, P., Dimitrova, R.: Issy：用于无限状态反应式系统规约与综合的综合工具。载于：Piskac, R., Rakamaric, Z.（编）《计算机辅助验证——第37届国际会议，CAV 2025，克罗地亚萨格勒布，2025年7月23日至25日，论文集，第四部分》。《计算机科学讲义》，第15934卷，第298–312页。施普林格（2025）。https://doi.org/10.1007/978-3-031-98685-7_14, https://doi.org/10.1007/978-3-031-98685-7_14[29] Heim, P., Dimitrova, R.: 面向高效无限状态反应式综合的时序逻辑转换。ACM编程语言论文集**9(POPL)**（2025年1月）。https://doi.org/10.1145/3704888, https://doi.org/10.1145/ 3704888[30] Hester, T., et al.: 基于演示的深度Q学习。载于：AAAI（2018）[31] Hussein, A., et al.: 模仿学习：学习方法综述。ACM计算调查（2017）[32] Kamradt, G.: Arc-agi-3预览：30天学习成果。https://arcprize.org/blog/arc-agi-3-preview-30-day-learnings（2025），访问日期：2025-02-22[33]Komatsu, K., Horita, H.: 通过轨迹示例为流程挖掘生成LTL公式。数据科学与智能系统期刊**3(4)**，304–311（2024年2月）。https://doi.org/10.47852/ bonviewJDSIS42022166, https://ojs.bonviewpress.com/index.php/ jdsis/article/view/2166[34] Lemieux, C.,Park, D.,Beschastnikh, I.: 通用LTL规约挖掘(t)。载于：2015年第30届IEEE/ACM自动化软件工程国际会议（ASE）。第81–92页（2015）。https://doi.org/10.1109/ ASE.2015.71[35] Muskardin, E., Aichernig, B.K., Pill, I., Pferscher, A., Tappler, M.: AALpy：主动自动机学习库。载于：自动化验证与分析技术国际研讨会。第67–73页。施普林格（2022）[36]Neider, D., Gavran, I.: 学习线性时序性质。CoRR**abs/1806.03953**（2018），http://arxiv.org/abs/1806.03953

[37] Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M.：亚马逊网络服务如何运用形式化方法。ACM通讯 **58**(4)，66–73（2015年4月）[38] Pnueli, A.：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）论文集。第46–57页。IEEE计算机学会（1977）[39] Pomerleau, D.A.：用于自主导航的人工神经网络高效训练（1991）[40] Raha, R., Roy, R., Fijalkow, N., Neider, D.：用于学习线性时序逻辑片段的可扩展任意时间算法（2024），https://arxiv.org/ abs/2110.06726[41] Rodr´ ıguez, A., Gorostiaga, F., S´anchez, C.：针对模理论线性时序逻辑的反例引导式反应式综合。载于：计算机辅助验证国际会议。第224–248页。施普林格（2025）[42] Rodr´ıguez, A., S´anchez, C.：模理论可实现性的布尔抽象。载于：计算机辅助验证国际会议。第305–328页。施普林格（2023）[43] Rodr´ ıguez, A., S´anchez, C.：模理论线性时序逻辑及有限迹线性时序逻辑的自适应反应式综合。载于：AAAI人工智能会议论文集。第38卷，第10679–10686页（2024）[44] Zhang, T., McCarthy, Z., Jow, O., Lee, D., Chen, X., Goldberg, K., Abbeel, P.：通过虚拟现实遥操作对复杂操作任务进行深度模仿学习。载于：IEEE机器人与自动化国际会议（ICRA）（2018）

定理 1 的证明。TSL$_f$ 的表达能力严格弱于 TSL。

证明。TSL$_f$ ⊆包含于TSL是显然的，因为TSL$_f$ 与采用有限轨迹语义的TSL使用相同的语法。我们通过给出一个TSL 规范来证明该包含关系是严格的：它在TSL中可实现，但在TSL$_f$中不可实现。

考虑以下 TSL 规范：

$$\mathbf{G}\left(\mathbf{F}\left[\mathtt{x} \leftarrow \mathtt{y}\right] \wedge \mathbf{F}\left[\mathtt{x} \leftarrow \mathtt{z}\right]\right).$$

该公式断言，在每个位置，最终都会发生一次将 y 更新为 x 的操作，以及一次将 z 更新为 x 的操作。

在 TSL 中可实现（无限轨迹）： 系统可以通过无限交替执行这两种更新来满足该公式。例如，以下无限计算模式：

$$[\mathtt{x} \leftarrow \mathtt{y}], [\mathtt{x} \leftarrow \mathtt{z}], [\mathtt{x} \leftarrow \mathtt{y}], [\mathtt{x} \leftarrow \mathtt{z}], \ldots$$

满足每个位置上的该公式，因为从任意位置出发，未来总会有一个位置发生每种更新。

在 TSL 中不可实现$_f$ （有限轨迹）： 反设某条长度为 n 、格式良好的有限轨迹 σ 满足该公式。那么：

1. 根据**G**的语义，该公式必须在每个位置成立，包括最终位置 $n-1$:

$$\sigma, n-1 \models \mathbf{F}\left[\mathtt{x} \leftarrow \mathtt{y}\right] \wedge \mathbf{F}\left[\mathtt{x} \leftarrow \mathtt{z}\right]$$

2. 根据 TSL 中**F** 的强语义$_f$，其中$\mathbf{F}\varphi \equiv \top\ \mathbf{U}\ \varphi$，我们有$\mathbf{F}\left[\mathtt{x} \leftarrow \mathtt{y}\right]$ 在位置 $n-1$ 要求：

$$\exists j \geq n-1.\, (j < n \wedge \sigma, j \models [\mathtt{x} \leftarrow \mathtt{y}])$$

唯一满足 $j \geq n-1$ 和 $j < n$ 的位置 j 是 $j = n-1$。因此：

$$\sigma, n-1 \models [\mathtt{x} \leftarrow \mathtt{y}]$$

3. 根据对称推理， σ， $n-1 \models [\mathtt{x} \leftarrow \mathtt{z}]$。4. 因此， $[\mathtt{x} \leftarrow \mathtt{y}] \in \sigma_{n-1}$ 和 $[\mathtt{x} \leftarrow \mathtt{z}] \in \sigma_{n-1}$二者均成立。这违反了良构性。

因此，没有任何良构有限轨迹能够满足此规约，这证明它在 TSL$_f$ 中不可实现。由于该规约在 TSL 中可实现，但在 TSL$_f$ 中不可实现，我们得出结论：TSL$_f$ $\subsetneq$ TSL。

B 轨迹生成

所有轨迹都以 jsonl 格式记录为状态快照序列，其中每个状态都是一个将变量名映射到整数值的字典。对于每个环境和训练规模，我们生成两组行为示范。正轨迹是到达目标状态的成功回合。对于基于网格的环境（FrozenLake、CliffWalking、 Taxi），我们使用广度优先搜索（BFS）计算最优路径，然后加入随机绕路，以增加轨迹多样性。负轨迹是失败回合，由随机游走和对正轨迹 BFS 搜索结果的偏离混合生成。对于Blackjack，我们使用基本策略（基于手牌点数的知名确定性策略）。负轨迹是不遵循该策略的轨迹。

冰冻湖。 一条从起点前往目标、避开洞的获胜轨迹。


```
{"x": 0, "y": 0, "goalx": 3, "goaly": 3, "h0x": 1, "h0y": 1, "h1x": 3, "h1y": 1, "h2x": 3, "h2y": 2}
{"x": 1, "y": 0, "goalx": 3, "goaly": 3, "h0x": 1, "h0y": 1, "h1x": 3, "h1y": 1, "h2x": 3, "h2y": 2}
{"x": 2, "y": 0, "goalx": 3, "goaly": 3, "h0x": 1, "h0y": 1, "h1x": 3, "h1y": 1, "h2x": 3, "h2y": 2}
{"x": 2, "y": 1, "goalx": 3, "goaly": 3, "h0x": 1, "h0y": 1, "h1x": 3, "h1y": 1, "h2x": 3, "h2y": 2}
{"x": 2, "y": 2, "goalx": 3, "goaly": 3, "h0x": 1, "h0y": 1, "h1x": 3, "h1y": 1, "h2x": 3, "h2y": 2}
{"x": 2, "y": 3, "goalx": 3, "goaly": 3, "h0x": 1, "h0y": 1, "h1x": 3, "h1y": 1, "h2x": 3, "h2y": 2}
{"x": 3, "y": 3, "goalx": 3, "goaly": 3, "h0x": 1, "h0y": 1, "h1x": 3, "h1y": 1, "h2x": 3, "h2y": 2}
```


悬崖行走。 一条在悬崖上方前往目标的获胜轨迹。


```
{"x": 0, "y": 0, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 0, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 1, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 2, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 3, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 4, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 5, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 6, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 7, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 8, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 9, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 10, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 11, "y": 1, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
{"x": 11, "y": 0, "goalx": 11, "goaly": 0, "cliffXMin": 1, "cliffXMax": 10, "cliffHeight": 1}
```


Taxi。 一条获胜轨迹：出租车在 RED 接乘客，并将其送至 DEST。


```
{"x": 2, "y": 2, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 1, "y": 2, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 0, "y": 2, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 0, "y": 1, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 0, "y": 0, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 1, "y": 0, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 2, "y": 0, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 3, "y": 0, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
{"x": 4, "y": 0, "DEST_x": 4, "DEST_y": 0, "PASS_x": 0, "PASS_y": 0, "RED_x": 0, "RED_y": 0, "GREEN_x": 4,
  "GREEN_y": 0, "YELLOW_x": 0, "YELLOW_y": 4, "BLUE_x": 3, "BLUE_y":4}
```


二十一点（基本策略）。11 点时要牌，面对弱庄家（明牌为 5）时 15 点停牌。


```
{"count": 11, "stood": false, "standThreshold": 17, "standVsWeakMin": 13, "isWeakDealer": true}
{"count": 15, "stood": false, "standThreshold": 17, "standVsWeakMin": 13, "isWeakDealer": true}
{"count": 15, "stood": true, "standThreshold": 17, "standVsWeakMin": 13, "isWeakDealer": true}
```

C 综合模板

列表 1.1：grid standard template.tsl。xMoves 和 yMoves 是我们 TSL$_f$ 挖掘过程中发现的、分别针对 x 和 y 的更新项析取式。 Taxi 模板包含额外的墙壁边界。

```
inBounds = (gte x B_MIN) && (lte x B_MAX) && (gte y B_MIN) && (lte y B_MAX)
          // && !(eq x w0x && eq y w0y) && !(eq x w00x && eq y w00y) {Taxi}
          // && !(eq x w1x && eq y w1y) && !(eq x w11x && eq y w11y) {Taxi}
xMoves = {discovered_x_updates}
yMoves = {discovered_y_updates}
assume {
    eq x START_X;
    eq y START_Y;
}
guarantee {
    G inBounds;                                          // Stay in bounds
    G ((xMoves && [y ← y])  ([x ← x] && yMoves));        // Discovered updates
    {mined_tslf_specification};                          // Mined Objective
}
```

同样，对于二十一点，该模板为可能的牌值设定了上下界，并将发到的牌作为系统的输入。系统控制着：

清单 1.2：blackjack.tsl

```
always assume {
    (gte handValue MIN_HAND) && (lte handValue MAX_HAND);
    (gte dealerCard MIN_DEALER) && (lte dealerCard MAX_DEALER);
}

guarantee {
    {mined_tslf_specification};
}
```

D 二十一点策略

1. 阈值策略。

$$\text{action}(c,d) = \begin{cases} \text{HIT} & \text{if } c < 17 \\ \text{STAND} & \text{if } c \geq 17 \end{cases}$$

2. 保守策略。

$$\text{action}(c,d) = \begin{cases} \text{STAND} & \text{if } c \geq 17 \\ \text{STAND} & \text{if } c \geq 12 \wedge 2 \leq d \leq 6 \\ \text{HIT} & \text{if } c \leq 11 \\ \text{HIT} & \text{otherwise} \end{cases}$$

3. 基本策略。

$$\text{action}(c,d) = \begin{cases} \text{STAND} & \text{if } c \geq 17 \\ \text{HIT} & \text{if } c \leq 11 \\ \text{STAND} & \text{if } c \geq 12 \wedge 2 \leq d \leq 6 \\ \text{HIT} & \text{otherwise} \end{cases}$$

E 评估训练时间

Game	方法	4	8	12	16	20	50	100	500	1000	
FrozenLake	TSL_f	3.1	2.2	3.0	3.8	5.0	–	–	–	–	
	Alergia	<.01	<.01	<.01	<.01	<.01	.02	.03	.18	.27	
	BehavClone	.23	.54	1.1	1.6	1.3	3.1	5.0	29	58	
	CART	<.01	<.01	<.01	<.01	<.01	<.01	<.01	.09	.13	
CliffWalking	TSL_f	14	8.9	9.8	12	11	–	–	–	–	
	Alergia	<.01	<.01	<.01	<.01	<.01	.05	.02	.09	.24	
	BehavClone	.84	.58	.83	1.2	1.3	3.6	8.4	45	98	
	CART	<.01	<.01	<.01	<.01	<.01	<.01	<.01	.06	.14	
Taxi	TSL_f	9.4	8.9	14	12	13	–	–	–	–	
	Alergia	<.01	<.01	<.01	<.01	<.01	.02	.04	.17	.37	
	BehavClone	1.2	.70	1.2	1.6	1.9	4.5	9.0	54	107	
	CART	<.01	<.01	<.01	<.01	<.01	<.01	.01	.29	.32	
Blackjack	TSL_f	.72	.34	.35	.35	1.2	–	–	–	–	
	Alergia	<.01	<.01	<.01	<.01	<.01	–	–	–	–	
	BehavClone	.74	.15	.16	.16	.35	–	–	–	–	
	CART	.36	<.01	<.01	<.01	<.01	–	–	–	–	

表 9：所有环境中的训练时间（秒）。Alergia 和 CART 的训练时间以毫秒计；BehavClone 随 n线性扩展。TSL$_f$ 需要 3–14 秒，但仅用少 1–2 个数量级的样本便实现了完美泛化。

我们报告了所有方法在各个环境中的训练时间（以秒为单位）。TSL$_f$ 函数发现与规约挖掘，以及其他基线学习，均在 Apple M1 Pro 处理器上本地进行。通过 Issy 进行的控制器综合则在 Modal 云基础设施上完成（x86-64、4 个 vCPU、4GB RAM）；FrozenLake、CliffWalking、Taxi 和二十一点的平均耗时分别为 21、38、29 和 3 分钟。