arXiv:2602.13222v1 [cs.CC] 26 Jan 2026

智能体系统的可计算性

预印本

Chatavut
Viriyasutheep.virie@gmail.com

2026年2月17日

摘要

本文介绍了 Quest Graph，这是一种用于分析有限上下文下智能体系统能力的形式化框架。我们定义了用于建模常见推理技术的抽象，并确立了它们的计算能力：基础 Quest Graph 等价于无限制图灵机；尽管正向有限任务决策过程（FQDP）应用广泛，但它仅等价于下推自动机（上下文无关）；而参考增强型 QDP（RQDP）只有在允许有状态查询时才能恢复图灵完备性。由于可计算性会影响效率，我们进而通过在计算图中模拟任务依赖项，分析了各模型的理论效率。我们表明，这一计算层级会转化为具体的性能权衡：与未增强（上下文无关）的系统相比，参考增强型（图灵完备）系统在模拟复杂图时的效率最高可提升至指数级。本文提供了一种形式化方法，用于分类和理解智能体系统的基本能力。

1 引言

智能体人工智能是一个新兴术语，指能够自主行动的系统（Durante 等人 [2024], Acharya 等人 [2025]）。这些系统只需极少的监督即可运行，能够维持自身目标，并策略性地利用可用资源来实现目标。语言模型（LM）是这类系统的核心，作为主要的决策模块，通过处理自然语言进行复杂推理和规划。

提升智能体系统性能有多种策略。最直接的方法是增大语言模型的参数规模，但这会增加训练和执行所需的计算量。另一种方法是扩展推理步骤，以涵盖多个视角（Wei 等人 [2022], Wang 等人 [2022], Liu 等人 [2023], Shinn 等人 [2023], Zhao 等人[2023], Wang 等人 [2024]）。这些技术需要越来越大的上下文来处理元推理。相反，更大的上下文并非总能带来更好的性能（Shi 等人 [2023], Liu 等人 [2024], Jiang 等人 [2024]），因为它可能引入无关信息，分散模型的注意力。事实上，另一类技术通过将复杂问题分解为更简单的层级子任务来提升性能，而这些子任务可以在较小的上下文大小下解决（Zhou 等人 [2022], Wang 等人 [2023], Chen 等人 [2024], Yao 等人 [2024], Besta 等人[2024], Ma 等人 [2025]）。这些技术使人们开始质疑仅通过增大上下文大小来提升性能的做法，并促使我们进行更根本的分析。

在这项工作中，我们提出一个问题：在上下文大小固定的情况下，系统的能力能达到什么程度？这与经典可计算性理论有关，该理论研究在计算资源有限的条件下能够计算什么（Hopcroft 等人 [2001]）。事实证明，在某些情境下，不同层次的可计算性不仅会影响能够计算什么，也会影响计算效率。

设想一组可重复执行的流程 A、B 和 C，其中 B 是异步的（Syme 等人 [2011]），且 C依赖于 B，而 B 依赖于 A。受限于正则表达式的系统只能按固定顺序处理 A、B 和 C，识别诸如 $(ABC)+$ 这样的模式。相比之下，具备上下文敏感可计算性的系统可以处理$A^nB^nC^n$ 模式。后者通过在继续执行C之前，对异步的 B 流程进行流水线处理，从而提高效率，并确保等待时间最短。在许多其他场景中，不同的可计算性层级也可能带来不同的效率权衡。

我们不再依赖示例场景，而是使用依赖图进行更形式化的分析。我们要探讨的问题是：给定一种期望的执行模式，具备不同计算能力的智能体系统能够多有效地模拟它？为回答这一问题，我们提出了一个名为 *Quest Graph* 的形式化框架，其名称源自智能体系统以目标为导向的特性。该框架通过展示智能体系统常见的推理模式对应于形式语言层级中的不同层级，对智能体系统进行分类（Chomsky [1956],Hopcroft 等人 [2001]）。随后，我们分析各类系统模拟计算图的效率。计算图是描述复杂计算关系的有力工具（Abadi 等人 [2016]）。我们认为，这项研究是迈向形式化理解智能体系统可计算性及其能力的基础性一步。

语言	自动机	智能体系统	推理特征	计算图复杂度
无限制	TM	Quest Graph	图遍历	$O(N^2)$
		RQDP, NRQDP	知识检索	$O(N^2 \log N)$
上下文无关（上下文无关语言）	PDA	NFQDP	搜索	$O(2^N)$
确定性上下文无关语言	DPDA	FQDP	层次结构	$O(2^N)$
正则	FSA	LM	自回归	n/a

表 1：主要结果总结：智能体架构与形式语言层级的对应关系。该表将每种系统变体映射到相应的自动机类别及其主要推理能力。最后一列报告了模拟含有 N 个节点的计算图（CG）所需的时间复杂度；标记为“n/a”的条目表示，在响应大小恒定的约束下，模型无法模拟该图。

2 有限上下文下的自回归语言模型

为了确立智能体可计算性的基准，我们首先考虑一种在有限上下文窗口上运行的标准语言模型（LM）——一种固定大小的内存缓冲区，其中包含模型可访问的近期词元序列。我们将这种语言模型定义为：根据该窗口的当前内容逐步生成下一个词元的模型。在每个计算步骤中，上下文会通过两种方式更新：模型自身的输出以自回归方式追加，任何新的外部输入也会加入其中。

这种抽象使我们能够分析语言模型的计算能力，而不必深入探讨可能影响模型可计算性的具体架构或训练方法。例如，循环神经网络和长短期记忆架构（Hochreiter 和 Schmidhuber [1997]）足以实现正则语言识别，而基于 Transformer 的模型则被证明能力较弱（Hahn [2020]）。

在此分析中，我们假 设模型的输出是确定性的，相当于将其采样温度设为零
现中（Vaswani 等人 [2017]）。
在许多基于能量的实

尽管有限上下文窗口语言模型表现出色，但它们在根本上仍存在局限。鉴于词元词汇表是有限的，可以形式化地证明：

理论 引理 1. 具有有限上下文的语言模型在计算上等价于有限状态机（有限状态机).

Del´etang 等人 [2022] 已证明，这种理论上的等价性也适用于自回归机制。这凸显了一个关键区别：有限上下文语言模型等价于有限状态机（Hopcroft 等人 [2001]），而其上下文无界的对应模型则是图灵完备的（P´erez 等人 [2021]）。尽管如此，我们仍在附录 E 中给出证明，以形式化地确立这一基准，并使本文内容完整自足。

语言模型的标准模型将决策模块和上下文缓冲区视为一个单一、整体的实体。我们提出将这些组成部分解耦，把推理模块表示为一个作用于外部记忆结构的纯函数。这种分离兼具实践和理论优势。在实践层面，它与现代无状态架构相契合，这类架构以可扩展性和可测试性著称（Fielding [2000]）。更重要的是，这种方法使我们能够定义一个计算能力不断增强的智能体系统层级，从而有效扩展复杂度层级。

3 QuestGraphs 和 AgentFunctions

“Quest Graph”这一概念的灵感来源于工作记忆模型，该模型描述了支持人类认知功能的组成部分（Baddeley [1992]）。工作记忆为决策等高级认知过程提供信息的临时存储空间。采用受任务启发的主题命名 convention，我们将其称为

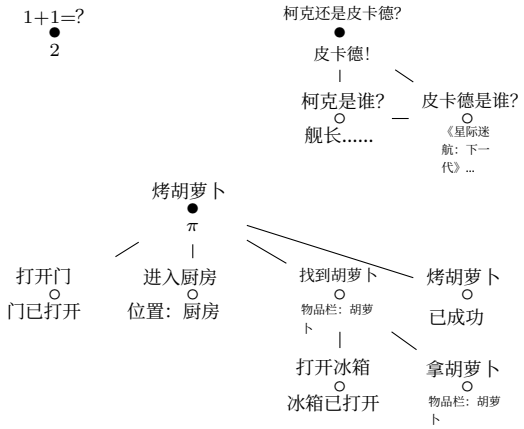


图 1：任务图示例。左上：单节点图。右上：表示多跳比较任务的图。下方：针对强化学习问题的两层分层图展开。焦点节点是带实心圆的节点。 π 表示父标记（定义见第 4 节）。

决策模块“智能体”。这两个组成部分之间的交互构成了我们提出的框架。

3.1 定义与机制

设 ***Quest Graph*** 是一个动态无向图 $\mathcal{Q} = (V,\ E)$。节点集 V 由有限节点配置集 $\mathcal{V}$ 中的元素组成。边集 E 连接成对的节点。Quest Graph 还保存对特定焦点节点 $v_f \in V$ 的引用。

每个节点配置 $v \in \mathcal{V}$ 都是一个元组 $(g,\ r)$，由两个组成部分构成：静态目标 $g \in \mathcal{G}$，在节点创建时设定且保持不变；以及动态响应 $r \in \mathcal{R}$，智能体可以在计算过程中对其进行修改。响应集 $\mathcal{R}$ 可以包含一个特殊的空符号 ε，用于表示未完成的状态。所有可能目标的集合 $\mathcal{G}$ 和所有可能响应的集合 $\mathcal{R}$ 都是有限的。

智能体通过读取焦点节点及其邻居的数据来操作 Quest Graph，以推断下一步动作，并将该图用作其专属工作记忆。

令 $C \in \mathbb{Z}^+$ 为一个正整数，用于定义智能体一次可考虑的最大邻居数量。该参数 C 并不限制节点在图中的实际度，而是限制智能体的同时处理容量，类似于认知负荷限制。令 $\mathcal{V}^{\leq C} = \bigcup_{k=0}^{C} \mathcal{V}^{1+k}$ 为一个有限集合，其中包括焦点节点的所有配置，以及其最多 C 个邻居所组成的有序集合。令 $\mathcal{E}$ 表示一个边指针集合，这些边指针指向焦点节点的某个邻居，或指向焦点节点上的一个自环。令 $\mathcal{E}^{\leq C} = \bigcup_{k=0}^{C} \mathcal{E}^{1+k}$ 为一个有限集合，其中包括指向焦点节点及其最多 C 个邻居的所有边指针配置。

接下来，我们定义智能体函数 χ。该函数接收 Quest Graph 的当前局部上下文——即 $\mathcal{V}^{\leq C}$中的一个元素，表示焦点节点及其邻居——并输出一个动作。共有三种可能的动作：1. 发现节点，创建一个具有指定目标的新节点，并将其连接到当前上下文中选定的节点。输出为 $\mathcal{V} \times \mathcal{E}^{\leq C}$中的一个元素。2. 响应后移动焦点，更新焦点节点的响应，并将焦点转移到其某个邻居。输出为 $\mathcal{R} \times \mathcal{E}$中的一个元素。3. 停止执行，终止计算且不返回任何参数。

通过这些动作，Quest Graph 的演化方式类似于玩家在角色扮演游戏中逐步完成任务。图2展示了这种演化的一个示例。

因此，这些机制引出了一个关键的理论问题：受有限上下文容量 C 限制的智能体，能否在该框架内实现通用计算？

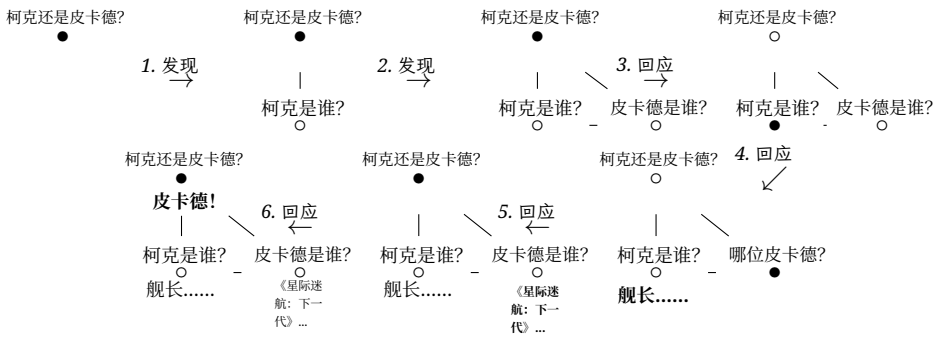


图2：多跳问答任务的 Quest Graph 演变过程。步骤1–2：“发现节点”动作生成子任务的新节点。步骤3：“响应后移动焦点”动作向主节点返回空更新，但将焦点转移到子任务。步骤4–6：后续的“响应后移动焦点”动作依次回答子任务，最终回答主任务节点。

3.2 图灵完备性

为确立我们框架的计算通用性，我们通过构造一个以 Quest Graph 作为运行环境、用于模拟标准图灵机（图灵机）（图灵 [1936], Hopcroft 等人 [2001]）的智能体函数，证明 Quest Graph 系统是图灵完备的。具体而言，我们考虑一台具有无限长纸带的图灵机，其头指针在每一步均可向左（L）或向右（R）移动。

定理2. 在有限上下文长度的 *Quest Graph* 上运行的智能体是图灵完备的。

我们的智能体函数是无状态的，完全依赖 Quest Graph 来存储记忆。因此，我们的构造类似于一种非常规图灵机变体。在这种模型中，状态并不保存在单独的内部寄存器中，而是直接编码在纸带上，纸带头也可以查看相邻单元格，以确定下一步动作。虽然文献中已有类似的非常规模型（Arulanandham [2007], Cantone 和 Cristofaro [2016]），但我们在此针对自身框架提供了证明概要。更详细的证明见附录A。

证明概要。 我们将图灵机的纸带表示为线性 Quest 图，其中每个节点对应一个纸带单元。每个节点的响应存储必要的图灵机配置：一个纸带符号、一个状态符号和一个方向符号（L 或 R）。模拟按步骤进行：智能体接收局部上下文，从中获取图灵机的当前配置，通过回应焦点节点写入下一配置，然后按照图灵机转移函数的指示移动焦点。此机制确保当前纸带符号始终存储在焦点节点的响应中，而图灵机的当前状态则存储在前一个节点的响应中；该节点是焦点节点的邻居之一。
关键在于，智能体可以通过检查方向符号，明确识别这个前一个节点。由于图灵机不能跳过单元格，且纸带无环，智能体可以使用一条简单规则：前一个单元格是方向符号与焦点节点方向符号相反的那个邻居。 □

尽管通用的 Quest Graph 框架是图灵完备的，但其不受限制的特性会带来实际挑战。重写任意节点响应的能力会导致状态空间组合爆炸，使得学习有效的智能体策略变得困难。此外，在必须保留清晰且不可篡改的动作因果关系记录以供学习的应用中，修改智能体的历史记录往往并不可取。

这促使我们对一种限制更多但又很常见的智能体推理模式进行建模。该模式由两条规则定义：智能体只能沿前向方向发现节点，且不能改写过去节点的响应。这些限制自然促使智能体采用层次化推理方法。为解决复杂问题，智能体必须将其分解为更简单的子任务，我们称之为子任务。这一过程会扩展一棵推理树：智能体在当前焦点处添加一个新的子节点（即子任务），并利用整个分支的上下文来指导下一步动作。这种层次化模式在许多智能体系统中普遍存在（Ahn 等人 [2022], Yao 等人 [2023], Schick 等人 [2023],Song 等人 [2023], Tang 等人 [2024]），也是分层强化学习（分层强化学习）的基石（Barto 和 Mahadevan [2003], Sutton 等人 [1999]）。

4 任务决策过程

我们提出任务决策过程（QDP），这是 Quest Graph 框架的一个限制更多的版本，专门用于对层次化推理进行建模。QDP 遵循四项关键约束：1. 它施加严格的结构规则，即新节点只能作为当前焦点节点的子节点与之相连；2. 发现操作分为两种不同类型。发现输入操作会创建一个新节点来存储外部信息，其目标是查询，响应则是收到的信息。发现子任务操作会创建一个目标明确、响应为空的新节点；随后，智能体将当前焦点节点标记为父节点，并立即将焦点转移到这个新的子节点；3. 智能体只有在完成当前任务后，才能“先响应，再返回”父节点；4. 只有当焦点节点为根节点时，智能体才能停止执行。图 1 中最下方的示意图展示了典型的 QDP 展开过程。

形式上，QDP 是部分可观测马尔可夫决策过程（POMDP）的一种变体，其定义元组包括状态空间（$\mathcal{V}$）、层次化动作空间（$\mathcal{G}$）、智能体函数（χ）、观测空间（Ω_{qdp}）和观测函数（$\mathcal{O}_{\text{qdp}}$）。$\mathcal{V}$ 和 $\mathcal{G}$ 分别对应 Quest Graph 中的节点集和目标集。具体而言，$\mathcal{G}$ 是原始 POMDP 动作空间与一组可发现的高层子任务目标的并集。同样，节点响应集（记作 $\mathcal{R}$）是原始 POMDP 观测空间、一组子任务结果，以及一个特殊配置符号 π 的并集，该符号表示任务是某个子任务的父节点。该结果集的构成可以灵活设计；例如，它可以包含离散结果，如{成功，失败，截断，未达成}，也可以包含其他形式的摘要信息。Quest Graph 中活跃节点的目标和响应共同构成了智能体决策的上下文。

虽然一般的 QDP 是图灵完备的，能力与无限上下文语言模型相当（P´erez 等人 [2021]），但我们的分析聚焦于一种限制更多的模型：有限任务决策过程（FQDP）。FQDP 限制每个节点只能有有限个子节点。一旦达到子节点数量上限，智能体便只能执行不会为焦点节点创建更多子节点的动作。这些约束从根本上降低了模型的计算能力。

定理 3. 有限任务决策过程（*FQDP*）等价于确定性下推自动机（*DPDA*）。

附录 B 中的详细证明通过在 FQDP 与确定性下推自动机（Hopcroft 等人 [2001]）之间建立双模拟来完成。这种等价关系直观易懂：FQDP 以层级化的后进先出方式管理子任务，这与栈在现代编程语言中处理函数调用的方式直接对应。

4.1 非确定性有限任务决策过程

FQDP 中的子任务机制鼓励对推理树进行确定性的深度优先探索。这种确定性的主要优势在于简化了策略学习，因为由此产生的执行轨迹是明确无歧义的。然而，另一种常见的模式是广度优先探索，即智能体并行创建多个子任务或解决方案，并利用外部机制对它们进行评估和选择（Yao 等人 [2024], Hao 等人[2023], Wang 等人 [2022]）。

我们将这一变体称为 非确定性有限任务决策过程 （NFQDP）。NFQDP 放宽了 FQDP 严格的子任务发现过程，将其进一步拆分为两个独立的动作：1. 发现子任务动作现在会创建一个新的子节点，而不会立即将焦点转移到该节点；2. 新增的追踪动作允许智能体将焦点转移到当前焦点节点的任意未完成邻居。

顾名思义，该模型在计算上等价于非确定性下推自动机（PDA）（Hopcroft 等人 [2001]）。发现动作与追踪动作的分离，使智能体能够遍历一棵预先构建的推理树。对于某些情况而言，这是必要的：单个任务配置可能会导向不同的推理路径，而这一差异在子任务被主动追踪之前，智能体无法确定。完整证明见附录 C。

以层级方式处理信息且具有有限上下文的系统属于这一类别。例如，ReAct 是一种 FQDP 系统，因为它以深度优先方式交替进行推理和工具调用（Yao 等人 [2023]）。分层强化学习智能体也符合这一模型：高层策略以嵌套方式为低层策略设定要实现的子目标（Sutton 等人 [1999], Barto 和 Mahadevan [2003]）。另一方面，思维树（Yao 等人 [2024]）和通过规划进行推理（Hao 等人 [2023]）等系统通过外部搜索机制探索不同的嵌套路径，因此可以建模为 NFQDP。

这些结果带来了一个重要启示：任何受 FQDP 或 NFQDP 约束运行的智能体系统都不是图灵完备的。相反，其计算能力受到限制，相当于能够识别上下文无关语言的机器。这种计算能力的下降凸显了对一种模型的需求，该模型

不仅要比下推自动机具有更强的表达能力，还要能够保留不可变的动作历史记录，同时在有限上下文窗口内运行。

5 参考增强型有限任务决策过程

受检索增强生成（RAG）系统中用于获取知识的检索机制启发（Lewis 等人 [2020]），我们提出了参考增强型有限任务决策过程（RQDP）。典型的 RAG 系统会查询外部知识库，以提升生成性能。然而，我们的机制利用检索来访问智能体自身过去的经验——这些节点无法通过局部上下文访问。这样既能访问无限的历史记录，又能确保智能体的展开过程保持不变，这是策略学习所期望的特性。

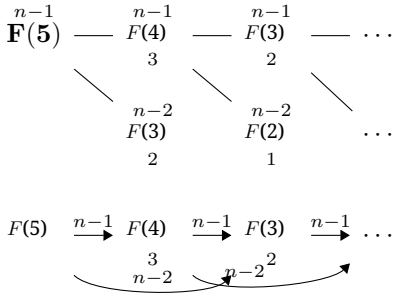


图 3：一个 RQDP 模拟斐波那契数的递归计算，$F(n) = F(n-1) + F(n-2)$。上：执行轨迹（展开过程），智能体以深度优先的方式探索依赖图，在检索 $F(n-2)$ 项的预计算值的同时，递归计算 $F(n-1)$ 项。为确保终止，我们假设在每个 $F(n-1)$ 节点都会发现一个省略的输入，以便倒数至基本情形。下：对应的引用图，其中的边由引用生成函数 τ 定义。每个节点中的标签（例如 $F(5)$）表示其引用；引用图中的相应节点存储缓存响应。

为了启用检索机制，系统需要一种区分节点的方法。每个节点都关联一个称为引用的标签，该标签取自一个无限集合 $\mathcal{T}$。该集合必须是无限的，才能容纳 Quest Graph 可能不断延伸的历史记录。如果引用集合是有限的，智能体理论上可以将整个引用配置编码到单个有限响应中，这并不会从根本上提升其计算能力，使其超越 FQDP。常见 RAG 系统中的无状态检索机制固有地受到这一限制，因为检索完全依赖于根据当前上下文生成的有限查询（Gao et al. [2023], Peng et al.[2024]）。

通过任何动作创建新节点时，Quest Graph 都会使用一个确定性的引用生成函数为其分配引用，$\tau : (\xi, g) \mapsto \xi'$。该函数将焦点节点的引用（$\xi \in \mathcal{T}$）和新节点的目标（$g \in \mathcal{G}$）映射为一个新引用 $\xi' \in \mathcal{T}$。

RQDP 引入了一种检索动作，定义为 $\mathcal{V}^{\leq C} \to \mathcal{G}$。当智能体选择此动作时，它会输出一个目标。随后，Quest Graph 会触发新节点的创建，这与发现操作类似。但 Quest Graph 不会立即将新节点的响应初始化为空，而是尝试在历史记录中搜索匹配的节点，并复制其响应。若历史节点满足以下条件，则视为匹配：1. 根据匹配算子 $\mathcal{T} \times \mathcal{T} \to \{\text{true}, \text{false}\}$，其引用与新节点的引用匹配；2. 在历史记录中所有匹配节点中，它是最近更新的节点。若未找到匹配项，则将检索节点的响应设为空符号 ε。

根据引用生成函数的构造，引用机制必须维护状态，以确保在无界历史记录中有足够的容量来区分引用。重要的是，这种设计并未违背智能体函数有限且无状态的假设。无界的引用标签和有状态的检索过程由 QuestGraph 管理，而呈现给智能体函数的局部上下文仍然是有限的，从而有效地将智能体与底层复杂性隔离开来。

引用机制可以通过单独实现一个引用图来实现，该图为每个唯一引用存储当前响应。在此实现中，引用图维护自己的头指针，用于追踪与当前焦点节点关联的引用，并随着焦点的变化而移动。新节点创建时会被添加到此图中，其连接关系由 τ 函数定义。当智能体回应主 Quest 图中的某个节点时，

系统会更新引用图中对应节点的响应。图 3 展示了 RQDP 的展开过程，其中底部示意图描绘了相应的引用图。

5.1 RQDP 的可计算性

RQDP 的构造在概念上与辅助下推自动机和栈自动机的构造相似。然而，这些模型通常受限于有界辅助记忆，因此分别属于对数空间可解的上下文无关语言类和多项式空间复杂度类（Sudborough [1978], Ginsburg 等人 [1967], Hopcroft 和 Ullman [1967]）。我们的模型在两个关键方面有所不同：其引用机制提供了更灵活的访问模式，并且允许历史记录无限增长。正是这些特性使 RQDP 达到了图灵完备性。

定理4. 参考增强型有限任务决策过程（*RQDP*）是图灵完 *te.*

证明概要。 我们将展示，RQDP 可以利用其引用机制表示图灵机（TM）的磁带位置，从而模拟标准图灵机。由于标准图灵机的纸带是单向无限的，这种模拟只需要一个能够递增引用的引用生成函数，这对应于磁头向右移动。完整证明见附录D。

该模拟将图灵机的纸带表示为两部分。头指针左侧的纸带部分由 RQDP 的类栈执行维护，其中，追踪子任务对应于一次“压入”。在此压入过程中，智能体会用父标记（π）覆盖父节点的响应，因此，这部分纸带的符号会直接存储在父节点的状态中。头指针右侧的纸带部分由引用机制管理，其中每个引用对应一个唯一的纸带单元，并在需要时检索其符号。

智能体的动作如下：1. 要向右移动，智能体会发现一个新的子节点，该节点由 τ 函数分配一个递增的引用（$\xi + 1$），从而将纸带向右延伸。要读取这一新位置的符号，它会使用检索操作。2. 要向左移动，智能体会完成当前节点，并将焦点移至其父节点，从而有效地“弹出”栈。这个新显现位置的纸带符号会直接从父节点的状态中读取。为了允许随后从这个“重新访问的”父节点向右移动，会使用一个“副本”节点，以防止形成循环，同时保留有限上下文约束。 □

此外，我们通过为 NFQDP 增加与 RQDP 相同的引用机制，定义了一个非确定性参考增强型有限任务决策过程（NRQDP）。由于 NRQDP 包含确定性 RQDP 的所有能力，因此它也具有图灵完备性。

引用机制使 RQDP 和 NRQDP 都能突破上下文无关语言的计算界限，同时保留执行轨迹以供学习。看来，这种机制是分层智能体达到至少上下文相关复杂度所必需的组成部分。

在确立了与形式语言层级中不同层次相对应的智能体自动机形式层次结构后，我们现在将焦点从纯粹的可计算性转向理论效率。为此，我们提出一个关键问题：在计算步骤方面，我们层次结构中的每个模型能以多高的效率模拟一般计算图?

6 模拟计算图

计算图是一种有向无环图（DAG），其中节点表示操作，边则定义它们之间的依赖关系。图的结构决定了执行顺序：只有在某个节点的所有前驱节点（即所有指向该节点的节点）都已计算完成后，才能计算该节点。

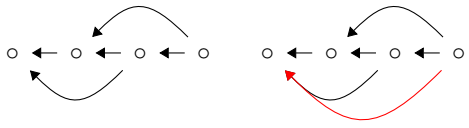


图4：计算图（左）及其最大依赖版本（右），两者的节点数量相同。

为分析 Quest Graph 及其变体在最坏情况下的效率，我们引入一种标准化基准：最大依赖计算图 （MCG）。MCG 是一种最大限度连接的有向无环图，对于给定的

节点排序，每个节点都依赖于所有排在它前面的节点。我们证明，任何计算图都是 MCG 的子图（证明见附录 F）。

这种方法无需依赖任何特定的图配置，即可进行一致的分析。一个包含 N 个节点的 MCG 恰有 $^{N}_{(N-1)\,2}$ 条边，为比较我们的智能体自动机的模拟效率提供了明确的依据。图4展示了一个计算图及其对应的 MCG 示例。

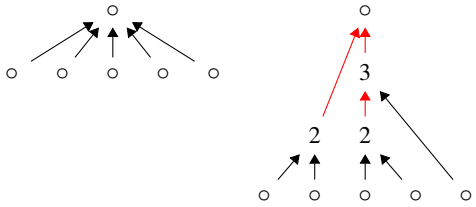


图 5：度为五的节点（左）及其对应的 $C = 2$ 叉代理节点树（右）。数字表示代理节点及其对响应大小可能造成的增长。红色边为新增的代理边。

当 MCG 的最大入度（即一个节点的依赖项数量）超过最大上下文大小 C 时，就会产生一个实际问题。由于 C 可能远小于节点总数 N，智能体无法一次性处理高度连接节点的所有依赖项。为解决这一问题，我们将图转换为一种适配智能体有限上下文的结构。任何入度大于 C 的节点都会被分解为一棵由代理节点构成的 k 叉树，其中树中的每个新节点最多有 C 个依赖项（图 5）。我们将所得结构称为有界最大依赖计算图 （BMCG）。

引理 5. 给定一个包含 N 个节点的最大依赖计算图（*MCG*），存在一个等价的有界最大依赖计算图（*BMCG*），由 O（N^2） 个节点和 O（N^2） 条边组成。

证明概要。 二次复杂度源于对所有节点的转换成本求和。对于任意单个节点，所需代理节点的数量与其“超额度数”成线性关系——也就是其入度超过上下文大小 C 的部分。在包含 N 个节点的 MCG 中，节点的入度线性递增，构成一个等差数列。因此，新增节点总数是这一线性序列中各超额度数之和，结果为二次总量，即 $O(N^2)$。由于每个新增代理节点也对应固定数量的新边，因此边的总数同样呈二次增长。附录 F 提供了完整的数值分析。 □

分解节点可能会产生代价，因为代理节点必须携带中间响应。k 叉树结构旨在确保这些信息在新节点之间均衡流动。然而，所需的响应大小完全取决于原始计算的性质。

在最极端的情况下，响应大小可能会显著增长，因为可以轻易地定义一个节点，使其计算需要了解整个前序图。对于不可压缩函数，代理节点可能只需将其子节点的响应拼接起来，从而使响应大小线性增长。相比之下，许多常见操作都具有很强的可压缩性。求和只需要用对数空间保存累计和，而布尔逻辑或求最大值等操作只需要常数空间。因此，这种转换导致的响应大小增长下界为 Ω(1)。

6.1 复杂度分析

现在，我们分析每种 Quest Graph 变体模拟计算图所需的时间复杂度。为此，我们首先确定智能体函数的计算成本。由于智能体在有限的局部上下文中运行，我们假设它能在常数时间内计算出输出， $O(1)$。

对于无限制 Quest Graph，可以在 Quest Graph 内预先构建 BMCG，并将根节点设为焦点节点。执行计算时，智能体可以按深度优先方式遍历该图，直到找到一个依赖项均已计算完成的节点。遍历图的时间复杂度受节点数量限制，因为对于每个非叶节点，智能体都必须先访问该节点，再回溯。

推论 6。 *Quest Graph* 可以通过 $O(N^2)$ 次操作模拟一个 *BMCG*。

与无限制 Quest Graph 不同，由于受到单父节点约束，NRQDP 无法直接实例化 BMCG。相反，NRQDP 必须将 BMCG 的探索树嵌入自身的任务图中，这一过程会

通常需要数量呈底层 MCG 大小指数级增长的节点。不过，引用机制允许 NRQDP 在这棵探索树的不同分支间共享某个节点的响应。具体做法是为初始 MCG 中的每个唯一节点分配一个唯一引用，从而大幅减少需要计算的节点数量。

确定性 RQDP 的运作方式有所不同，因为其动作集限制更多，无法在预先构建的探索树中导航。相反，必须通过发现输入动作将节点的依赖项提供给智能体。这迫使智能体在执行过程中动态发现图的结构，同时利用其引用机制检索先前计算出的节点。

对于 NRQDP 和 RQDP，模拟均以记忆化深度优先遍历的方式进行。为了计算任意给定节点，智能体会使用各依赖项的引用来检索它们。如果检索到的依赖项响应为空，表明该依赖项尚未计算，智能体就会先递归地对该依赖项进行模拟。为确保所有依赖项均已处理，智能体需要检查底层 MCG 的每一条边。由于总复杂度等于边的数量乘以每次检索操作的成本，因此我们可以得出：

推论 7。 *NRQDP* 和 *RQDP* 可以用 $O(N^2 \log N)$ 次操作模拟 *BMCG*。

在此，我们假设检索操作的时间复杂度相对于活跃引用的数量呈对数级，$O(\log N)$。这反映了在无限集合 $\mathcal{T}$ 中查找匹配引用所固有的成本。实际应用中，引用机制可以通过键值存储实现，而这通常意味着对数级的访问时间（Kleppmann [2017]）。另一种实现方式是使用单独的引用图（如前文所述），此时对数级成本并非发生在生成引用时，而是在响应后移动动作中查找正确的父节点引用并返回时产生。这两种实现策略的渐进复杂度相同。

NFQDP 和 FQDP 会动态遍历图，与加入引用机制的对应系统类似，但它们无法引用过去的结果。由于缺少这一能力，智能体每次需要共享依赖项时都必须重新计算。由于响应大小常数下界不允许它们将大型子树的结果直接存储在响应字段中，因此必须进行这种重新计算。

推论 8. *NFQDP* 和 *FQDP* 可以用 $O(2^N)$ 次操作模拟 *BMCG*。

证明. 计算依赖子树的代价由两部分组成：节点的直接代理树内部的计算，以及对其底层依赖项的递归计算。代理树的代价与其代理节点数量成正比，而代理节点数量又与该节点的入度和上下文大小 C 之差成正比。

令 $S(N)$ 表示计算一个具有 $N-1$ 个依赖项的节点所需的操作数。我们可以通过递推关系为此代价确定下界和上界。下界为 $S^-(N) = 1 + \sum_{i=1}^{N-1} S^-(i)$，其解为 $S^-(N) = 2^{N-1}$。上界为 $S^+(N) = N + \sum_{i=1}^{N-1} S^+(i)$，其解为$S^+(N) = 2^N - 1$。 $S^-(1) = S^+(1) = 1$。

由于下界 和上界均为 $O(2^N)$，因此需要重新计算的子树大小被限制在 $O(2^N)$。 □

最后，标准语言模型从根本上受到更多限制。与提供结构化图状记忆的 Quest Graph 各种变体不同，语言模型只能使用大小固定为 C 的简单线性上下文窗口。要模拟 BMCG，语言模型必须将先前计算出的中间结果逐一存储在这个单一窗口中。随着依赖规模增大，必须存储的中间结果数量会随 k 叉代理树的高度增长，即按 N 的对数因子增长，最终超过 C。

推论 9. 在响应大小受常数下界限制时，标准语言模型无法模拟任意 *BMCG*。

至此，我们完成了对 Quest Graph 及其各种变体在模拟计算图这一背景下的分析。

7 结论

在本研究中，我们提出了 Quest Graph，这是一种用于分析具有有限上下文窗口的智能体系统计算能力的理论框架。我们证明，该框架中旨在模拟常见推理模式的不同架构变体，分别对应形式语言层级中的不同层次：1. 用于图遍历的无限制 Quest 图是图灵完备的。2. 有限任务决策过程（FQDP）和非确定性

用于层次化推理的 FQDP（NFQDP）分别等价于确定性和非确定性下推自动机。3. 参考增强型 FQDP（RQDP）和非确定性 RQDP（NRQDP）在配备能够维持状态的引用机制时，同样是图灵完备的。这确立了智能体的设计与其根本表达能力之间的明确联系。

此外，我们分析了这些模型在模拟一般计算图时的理论效率，并得出了以下时间复杂度：1. 无约束 Quest 图可以在 $O(N^2)$ 次操作内模拟有界最大依赖计算图（BMCG）。2. RQDP 和 NRQDP 可以在 $O(N^2 \log N)$ 次操作内模拟 BMCG。3. FQDP 和 NFQDP 可以在 $O(2^N)$ 次操作内模拟 BMCG。4. 当响应大小受限于常数时，标准语言模型无法模拟任意 BMCG。我们的研究结果总结见表 1。

8 语言模型使用声明

在撰写本文的过程中，我们使用人工智能语言模型提供辅助，仅用于最终文本润色和文献检索。这一决定源于我们的信念：智识发现的过程本身就是科学事业中令人欣喜且不可或缺的一部分。我们声明，所有核心智识贡献、概念构思以及对本文的最终责任，均完全由人类作者承担。

引用

Mart´ın Abadi、Ashish Agarwal、Paul Barham、Eugene Brevdo、Zhifeng Chen、Craig Citro、Greg S Corrado、Andy Davis、Jeffrey Dean、Matthieu Devin 等。《Tensorflow：异构分布式系统上的大规模机器学习》。*arXiv*预印本 *arXiv:1603.04467*，2016年。Deepak Bhaskar Acharya、Karthigeyan Kuppan 和 B Divya。《智能体人工智能：面向复杂目标的自主智能——一项全面综述》。*IEEE Access*，2025年。Michael Ahn、Anthony Brohan、Noah Brown、Yevgen Chebotar、Omar Cortes、Byron David、Chelsea Finn、Keerthana Gopalakrishnan、Karol Hausman、Alex Herzog 等。《按我能做的去做，而不是按我说的去做：以机器人可供性为自然语言奠基》。*arXiv*预印本 *arXiv:2204.01691*，2022年。Joshua J Arulanandham。《非常规"无状态"图灵式机器》。载于非常规计算国际会议，第55–61页。施普林格，2007年。Alan Baddeley。《工作记忆》。*Science*，255(5044)：556–559，1992年。Andrew G Barto 和 Sridhar Mahadevan。《分层强化学习的最新进展》。离散事件动态系统，13(4)：341–379，2003年。Maciej Besta、Nils Blach、Ales Kubicek、Robert Gerstenberger、Michal Podstawski、Lukas Gianinazzi、Joanna Gajda、Tomasz Lehmann、Hubert Niewiadomski、Piotr Nyczyk 等。《思维图：利用大语言模型解决复杂问题》。载于*AAAI*人工智能会议论文集，第38卷，第17682–17690页，2024年。Domenico Cantone 和 Salvatore Cristofaro。《一种没有控制状态的图灵机变体及其与有界时序记忆的联系》。载于*ICTCS*，第36–48页，2016年。Sijia Chen、Baochun Li 和 Di Niu。《思维增强：利用大语言模型进行试错式问题求解》。*arXiv*预印本 *arXiv:2402.11140*，2024年。诺姆·乔姆斯基。《语言描述的三种模型》。*IRE*信息论汇刊，2(3)：113–124，1956年。诺姆·乔姆斯基。《关于文法的某些形式属性》。信息与控制，2(2)：137–167，1959。Gr´egoire Del´etang、Anian Ruoss、Jordi Grau-Moya、Tim Genewein、Li Kevin Wenliang、Elliot Catt、Chris Cundy、Marcus Hutter、Shane Legg、Joel Veness 等。《神经网络与乔姆斯基层级》。*arXiv*预印本 *arXiv:2207.02098*，2022年。Zane Durante、Qiuyuan Huang、Naoki Wake、Ran Gong、Jae Sung Park、Bidipta Sarkar、Rohan Taori、Yusuke Noda、Demetri Terzopoulos、Yejin Choi 等。《智能体人工智能：多模态交互前景综述》。*arXiv*预印本 *arXiv:2401.03568*，2024年。Roy Thomas Fielding。《架构风格与基于网络的软件架构设计》。加利福尼亚大学尔湾分校，2000年。

Yunfan Gao、Yun Xiong、Xinyu Gao、Kangxiang Jia、Jinliu Pan、Yuxi Bi、Yixin Dai、Jiawei Sun、Haofen Wang 和 Haofen Wang。《大语言模型的检索增强生成：综述》。*arXiv*预印本 *arXiv:2312.10997*，2(1)，2023年。Seymour Ginsburg、Sheila A Greibach 和 Michael A Harrison。《栈自动机与编译》。美国计算机协会期刊（*JACM*），14(1)：172–201，1967年。Michael Hahn。《神经序列模型中自注意力的理论局限》。计算语言学协会会刊，8：156–171，2020年。Shibo Hao、Yi Gu、Haodi Ma、Joshua Hong、Zhen Wang、Daisy Wang 和 Zhiting Hu。《用语言模型进行推理，就是用世界模型进行规划》。载于*2023*年自然语言处理经验方法会议论文集，第8154–8173页，2023年。Sepp Hochreiter 和 J¨urgen Schmidhuber。《长短期记忆》。神经计算，9(8)：1735–1780，1997年。John E. Hopcroft 和 Jeffrey D. Ullman。《非擦除栈自动机》。计算机与系统科学期刊，1(2)：166–186，1967年。John E Hopcroft、Rajeev Motwani 和 Jeffrey D Ullman。《自动机理论、语言与计算导论》。*ACM SIGACT*新闻，32(1)：60–65，2001年。Zhouyu Jiang、Mengshu Sun、Lei Liang 和 Zhiqiang Zhang。《检索、总结、规划：以迭代方法推进多跳问答》。*arXiv*预印本 *arXiv:2407.13101*，2024年。Martin Kleppmann。《设计数据密集型应用：可靠、可扩展且可维护系统背后的重要理念》。“O’Reilly Media, Inc.”，2017年。Patrick Lewis、Ethan Perez、Aleksandra Piktus、Fabio Petroni、Vladimir Karpukhin、Naman Goyal、Heinrich K¨uttler、Mike Lewis、Wen-tau Yih、Tim Rockt¨aschel 等。《面向知识密集型自然语言处理任务的检索增强生成》。神经信息处理系统进展，33：9459–9474，2020年。Nelson F Liu、Kevin Lin、John Hewitt、Ashwin Paranjape、Michele Bevilacqua、Fabio Petroni 和 Percy Liang。《迷失在中间：语言模型如何利用长上下文》。计算语言学协会会刊，12：157–173，2024年。Tengxiao Liu、Qipeng Guo、Yuqing Yang、Xiangkun Hu、Yue Zhang、Xipeng Qiu 和 Zheng Zhang。《规划、验证与切换：通过多样化思维链整合推理》。*arXiv*预印本 *arXiv:2310.14628*，2023年。Weiyu Ma、Qirui Mi、Yongcheng Zeng、Xue Yan、Runji Lin、Yuqiao Wu、Jun Wang 和 Haifeng Zhang。《大语言模型玩星际争霸II：基准测试与摘要链方法》。神经信息处理系统进展，37：133386–133442，2025年。Boci Peng、Yun Zhu、Yongchao Liu、Xiaohe Bo、Haizhou Shi、Chuntao Hong、Yan Zhang 和 Siliang Tang。《图检索增强生成：综述》。*arXiv*预印本 *arXiv:2408.08921*，2024年。Jorge P´erez、Pablo Barcel´ó 和 Javier Marinkovic。《注意力是图灵完备的》。机器学习研究期刊，22(75)：1–35，2021年。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》。神经信息处理系统进展，36：68539–68551，2023年。Freda Shi、Xinyun Chen、Kanishka Misra、Nathan Scales、David Dohan、Ed H Chi、Nathanael Sch¨arli 和 Denny Zhou。《大语言模型很容易被无关上下文分散注意力》。载于国际机器学习会议，第31210–31227页。PMLR，2023年。Noah Shinn、Federico Cassano、Ashwin Gopinath、Karthik Narasimhan 和 Shunyu Yao。《Reflexion：采用言语强化学习的语言智能体》。神经信息处理系统进展，36：8634–8652，2023年。Chan Hee Song、Jiaman Wu、Clayton Washington、Brian M Sadler、Wei-Lun Chao 和 Yu Su。《LLM-Planner：利用大语言模型为具身智能体进行少样本具身规划》。载于*IEEE/CVF*国际计算机视觉会议论文集，第2998–3009页，2023年。Ivan Hal Sudborough。《确定性上下文无关语言的带复杂度》。美国计算机协会期刊（*JACM*），25(3)：405–414，1978年。Richard S Sutton、Doina Precup 和 Satinder Singh。《马尔可夫决策过程与半马尔可夫决策过程之间：强化学习中时间抽象的框架》。人工智能，112(1-2)：181–211，1999年。Don Syme、Tomas Petricek 和 Dmitry Lomov。《F#异步编程模型》。载于声明式语言实用方面国际研讨会，第175–189页。施普林格，2011年。

Yimin Tang、Yurong Xu、Ning Yan 和 Masood Mortazavi。通过内循环查询机制提升大语言模型的长上下文性能。*arXiv* 预印本 *arXiv:2410.12859*，2024年。A Turing。论可计算数及其在判定问题中的应用。《伦敦数学学会会刊》*Series/2 (42)*，第 230–42 页，1936年。Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N Gomez、Łukasz Kaiser 和 Illia Polosukhin。注意力机制就是你所需要的一切。神经信息处理系统进展，第 30 卷，2017年。Jianing Wang、Qiushi Sun、Xiang Li 和 Ming Gao。通过知识链提示提升语言模型的推理能力。*arXiv* 预印本 *arXiv:2306.06427*，2023年。Lei Wang、Chen Ma、Xueyang Feng、Zeyu Zhang、Hao Yang、Jingsen Zhang、Zhiyuan Chen、Jiakai Tang、Xu Chen、Yankai Lin 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)：186345，2024年。Xuezhi Wang、Jason Wei、Dale Schuurmans、Quoc Le、Ed Chi、Sharan Narang、Aakanksha Chowdhery 和 Denny Zhou。自洽性提升语言模型中的思维链推理。*arXiv* 预印本 *arXiv:2203.11171*，2022年。Jason Wei、Xuezhi Wang、Dale Schuurmans、Maarten Bosma、Fei Xia、Ed Chi、Quoc V Le、Denny Zhou 等。思维链提示能够激发大语言模型的推理能力。神经信息处理系统进展，第 35 卷：24824–24837，2022年。Shunyu Yao、Jeffrey Zhao、Dian Yu、Nan Du、Izhak Shafran、Karthik Narasimhan 和 Yuan Cao。ReAct：融合语言模型中的推理与行动。载于国际学习表征会议（*ICLR*），2023年。Shunyu Yao、Dian Yu、Jeffrey Zhao、Izhak Shafran、Tom Griffiths、Yuan Cao 和 Karthik Narasimhan。思维树：利用大语言模型进行审慎问题求解。神经信息处理系统进展，第 36 卷，2024年。Wayne Xin Zhao、Kun Zhou、Junyi Li、Tianyi Tang、Xiaolei Wang、Yupeng Hou、Yingqian Min、Beichen Zhang、Junjie Zhang、Zican Dong 等。大语言模型综述。*arXiv* 预印本 *arXiv:2303.18223*，1(2)，2023年。Denny Zhou、Nathanael Sch¨arli、Le Hou、Jason Wei、Nathan Scales、Xuezhi Wang、Dale Schuurmans、Claire Cui、Olivier Bousquet、Quoc Le 等。从少到多提示能够实现大语言模型中的复杂推理。*arXiv* 预印本 *arXiv:2205.10625*，2022年。

一种任务图的可计算性

定义 10。 ***Quest Graph*** 是由有限个组成部分构建而成的动态无向图。这些组成部分如下：

- 所有可能目标的有限集合， $\mathcal{G}$。
- 所有可能响应的有限集合， $\mathcal{R}$。响应集可以包含一个特殊的空符号 ε，以表示不完整 状态。

节点配置是一个元组 (g, r)，其中 $g \in \mathcal{G}$ 且 $r \in \mathcal{R}$。$\mathcal{G}$ 和 $\mathcal{R}$ 都是有限集合，因此所有可能的节点配置集合 $\mathcal{V} = \mathcal{G} \times \mathcal{R}$ 也是有限的。

Quest Graph 的一个具体实例是一个元组 $\mathcal{Q} = (V, E)$，其中：

- V 是图中当前已实例化的节点集合。每个节点 $v \in V$ 都具有来自集合 $\mathcal{V}$ 的配置。
- E 是 V 中节点之间的无向边集合。

该图维护着对特定焦点节点 $v_f \in V$ 的引用。

一个局部上下文 由焦点节点及其在固定容量 $C \in \mathbb{Z}^+$ 范围内的邻居组成。所有可能的局部上下文集合记为 $\mathcal{V}^{\leq C} = \bigcup_{k=0}^{C} \mathcal{V}^{1+k}$。令 $\mathcal{E}$ 表示一组边指针，这些边指针指向焦点节点的某个邻居，或指向焦点节点上的一个自环。所有可能的边指针集合记为 $\mathcal{E}^{\leq C} = \bigcup_{k=0}^{C} \mathcal{E}^{1+k}$。

智能体函数 χ接收一个局部上下文，并执行以下三种可能的动作之一：

- 发现节点：返回一个新节点，以及一组指向将与该新节点连接的邻居的边指针。此输出是 $\mathcal{V} \times \mathcal{E}^{\leq C}$ 的一个元素，其中 $\mathcal{E}^{\leq C}$ 是边指针集。
- 响应并移动焦点：返回当前节点的响应，以及一个指明下一个焦点节点的边指针。此输出是 $\mathcal{R} \times \mathcal{E}$ 的一个元素。
- 停止：不返回任何内容，并终止计算。

设一台图灵机由七元组 $(S_{\text{tm}}, \Gamma_{\text{tm}}, \Sigma_{\text{tm}}, \delta_{\text{tm}}, s^0_{\text{tm}}, b_{\text{tm}}, F_{\text{tm}})$ 定义。对于我们的模拟，关键组成部分是有限状态集 (S_{tm})、纸带字母表 (Γ_{tm})、初始状态 $(s^0_{\text{tm}} \in S_{\text{tm}})$、转移函数 $\delta_{\text{tm}} : S_{\text{tm}} \times \Gamma_{\text{tm}} \to S_{\text{tm}} \times \Gamma_{\text{tm}} \times \{L, R\}$，以及接受状态集 $F_{\text{tm}} \subseteq S_{\text{tm}}$。其余组成部分——输入字母表 $\Sigma_{\text{tm}} \subseteq \Gamma_{\text{tm}}$ 和空白符号 $b_{\text{tm}} \in \Gamma_{\text{tm}} \setminus \Sigma_{\text{tm}}$——属于标准定义，但并非模拟机制的核心。

定理 11. 存在一个上下文容量为 $C = 2$ 的智能体函数，它通过在 ***Quest Graph*** 上运行，可以模拟任意图灵机。

证明. 我们将图灵机的纸带表示为线性 Quest 图，其中每个节点对应一个纸带单元，且最多有两个邻居（即相邻单元格）。每个节点的目标组件均未使用；所有必要的图灵机信息都编码在响应字段中。

我们将可能的响应集合 $\mathcal{R}$ 定义为所有图灵机纸带配置，再加上表示纸带两端的两个特殊符号： $\mathcal{R} = (S_{\text{tm}} \times \Gamma_{\text{tm}} \times \{L, R\}) \cup \{\varepsilon_L, \varepsilon_R\}$。因此，标准响应包含一个状态、一个纸带符号和一个方向。特殊响应 $\varepsilon_d = (\varepsilon, \varepsilon, d)$ 用于标记动态创建的纸带边界，其中 $d \in \{L, R\}$。

初始图灵机纸带配置在线性 Quest Graph 节点链上，焦点节点代表图灵机读写头。设置过程分为三步：

1. 将初始纸带符号写入每个对应节点的响应字段。2. 接下来，初始化方向符号，使其指向焦点内侧：左侧的每个节点都将方向设为 R，右侧的每个节点都将方向设为 L。3. 最后，将图灵机的初始状态 s^0_{tm} 放置在与焦点相邻的单个节点上（例如左侧邻居）。为确保智能体能够识别该节点，将焦点节点自身的方向设为与该邻居的方向相反。所有其他节点上的状态字段均可任意设置。例如，一个有效的初始配置为： $[(\cdot, \cdot, R), (s^0_{\text{tm}}, \cdot, R), (\cdot, \cdot, L), (\cdot, \cdot, L), (\cdot, \cdot, L)]$ 其中中间的节点是焦点，圆点表示通配符。

在模拟过程中，智能体的函数会在每一步执行。它读取自己的局部上下文（焦点节点及其一个或两个邻居），并根据以下两种情况确定适当的动作：

- 首先，如果智能体位于纸带的一端（即焦点节点的度为 1），其任务就是延伸纸带。它执行发现操作，创建一个目标为空的新节点，并赋予该节点一个特殊响应 ε_d，其中 d 是与现有邻居方向相反的方向。

- 否则，如果焦点节点位于纸带中间，智能体将按以下方式模拟标准图灵机转移：

  1. 读取状态和符号：智能体从焦点节点的响应中读取当前纸带符号。然后，它通过一个简单规则识别前一个节点：前一个节点是方向符号与焦点节点自身方向符号相反的邻居。随后，智能体从该前一个节点的响应中读取图灵机的当前状态。2. 计算并写入：智能体根据当前状态和纸带符号，模拟图灵机的转移函数 δ_{tm}，以获得新状态、新符号和新的移动方向。然后，它用这三个新值更新焦点节点的响应。3. 移动：最后，智能体执行移动焦点操作，将焦点移至新移动方向所指示的邻居节点。

这一过程会不断重复，直到智能体读取到图灵机接受状态集合中的一个状态，此时模拟停止。智能体的完整逻辑已形式化表示于表2中。

To 为证明模拟的正确性，我们展示这些不变模式始终成立：

- 当前纸带符号可以从焦点节点的响应中识别出来。这一点得以保证，是因为智能体会在移动之前将新的纸带符号写入焦点节点。
- 图灵机的当前状态始终存储在焦点节点某个邻居的响应中，并且可以利用焦点节点上的方向符号明确识别。这是因为局部上下文 [左邻居、焦点节点、右邻居] 中的方向符号配置始终符合以下两种模式之一：$[R, L, L]$ 或 $[R, R, L]$。这种可预测性源于图灵机读写头每次只移动一个单元格，且纸带不是循环的这一事实。如果焦点节点的响应具有方向 L，这意味着上次访问该节点时以向左移动结束；进而说明智能体必定是从右侧到达的，反之亦然。因此，前一个节点始终是方向符号与焦点节点相反的那个邻居。

- 每个节点的度最多为二，从而确保纸带保持线性。这是因为智能体只有在纸带的一端时才会创建新节点。

鉴于 S_{tm} 和 Γ_{tm} 都是有限的，参与图灵机模拟的每个 Quest Graph 节点最多与两个邻居交互，并且智能体是无状态的（按照表2，通过焦点节点及其两个邻居从 Quest Graph 中获取所有必要信息），这一构造忠实地模拟了任意图灵机。因此，采用这种智能体函数的 Quest Graph 框架是图灵完备的。 □

v_f，邻居 1，邻居 2	返回动作
$(\cdot, t, R), (s^1, \cdot, R), (s^2, \cdot, L)$	响应 $\delta_{\text{tm}} : (s^2, t) \mapsto (s', t', d')$ 并移动 d'
$(\cdot, t, L), (s^1, \cdot, R), (s^2, \cdot, L)$	响应 $\delta_{\text{tm}} : (s^1, t) \mapsto (s', t', d')$ 并移动 d'
$(\cdot, t, R), (s^1, \cdot, L), (s^2, \cdot, R)$	响应 $\delta_{\text{tm}} : (s^1, t) \mapsto (s', t', d')$ 并移动 d'
$(\cdot, t, L), (s^1, \cdot, L), (s^2, \cdot, R)$	响应 $\delta_{\text{tm}} : (s^2, t) \mapsto (s', t', d')$ 并移动 d'
$(\cdot, t, R), (\cdot, \cdot, R), (f, \cdot, L)$	stop
$(\cdot, t, L), (f, \cdot, R), (\cdot, \cdot, L)$	stop
$(\cdot, \cdot, \cdot), (\cdot, \cdot, R)$, -	通过响应 ε_L 发现并链接到焦点节点
$(\cdot, \cdot, \cdot), (\cdot, \cdot, L)$, -	通过响应 ε_R 进行发现并链接到焦点节点

表2：用于模拟任意图灵机的智能体函数。点号（$\cdot$）表示通配符，变量定义如下：$s^1, s^2, s' \in S_{\text{tm}}$；$t, t' \in \Gamma_{\text{tm}}$；$d' \in \{L, R\}$；以及 $f \in F_{\text{tm}}$。

虽然这一证明采用了一种专门的 Quest Graph，其中节点最多有两个邻居，但实际应用很可能会涉及连接度更高的图。这种更丰富的连接结构并不会削弱该框架的

表达能力，前提是智能体仍能模拟基础的 $C = 2$ 情形（例如，在更复杂的图中聚焦于一条“类似纸带”的路径）。

B 任务决策过程的可计算性

我们正式定义任务决策过程（QDP）及其有限版本。

定义 12. 一个任务决策过程 （*QDP*）是 *Quest Graph*（附录*A*）的一种约束更强的变体，被形式化为一个类似于部分可观测马尔可夫决策过程的元组：$(\mathcal{V}_{qdp}, \mathcal{G}, \chi, R_{qdp}, \Omega_{qdp}, \mathcal{O}_{qdp}, \gamma)$。与标准部分可观测马尔可夫决策过程的关键区别在于，*QDP* 以确定性方式运行。它不使用概率转移函数，而是使用确定性智能体函数 χ，该函数将局部上下文直接映射到一个动作。各组成部分定义如下：

- $\mathcal{V}_{qdp}$：有限状态集，对应于一组增广的 *Quest Graph* 节点。
- $\mathcal{G}$：一个有限的动作集，对应于 *Quest Graph* 中的目标集。
- χ：确定性智能体函数，用于取代标准转移函数。
- R_{qdp},Ω_{qdp} $\mathcal{O}_{qdp}$ γ：分别为标准奖励函数、观测集、观测函数和折扣因子。

每个任务节点 $v \in \mathcal{V}_{qdp}$ 是一个元组(g, r)，其中 $g \in \mathcal{G}$ 是目标，响应为 $r \in \mathcal{R} \cup \{\pi\}$。响应集扩充了一个特殊的父标记， π。带有此标记的节点是当前焦点节点的父节点。为简洁起见，我们将 *QDP* 状态集记为 $\mathcal{V}$。

QDP 通过四种主要的分层动作运行：

- 发现输入：添加一个新的输入节点作为当前焦点节点的子节点。
- 发现子任务：将一个新的任务节点添加为当前焦点节点的子节点，将当前焦点节点标记为父节点，并将焦点转移到新的子节点。
- 完成任务：使用响应后移动动作记录结果，并将焦点移回其父节点。
- 停止：终止计算。只有当焦点节点没有父节点时，才允许执行此动作。

一个有限 ***QDP*** （*FQDP*）是一种 *QDP*，它进一步限制每个节点只能有有限个子节点。达到这一限制后，智能体只能执行不会为焦点节点创建额外子节点的动作。

通过以这种方式限制智能体，FQDP 不再是图灵完备的。相反，它在计算上等价于一种能力较弱但仍然重要的计算模型：

定理 13. 有限任务决策过程（*FQDP*）在计算上等价于确定性下推自动机（*DPDA*）。

将下推自动机（PDA）定义为一个七元组 $(S_{\text{pda}}, \Sigma_{\text{pda}}, \Gamma_{\text{pda}}, \delta_{\text{pda}}, s^0_{\text{pda}}, Z^0_{\text{pda}}, F_{\text{pda}})$，其中：

- S_{pda} 是一个有限状态集。
- Σ_{pda} 是一个有限的输入符号集合（即输入字母表）。
- Γ_{pda} 是一个有限的栈符号集合（即栈字母表）。
- δ_{pda} 是转移函数。
- ${s^0}_{\text{pda}} \in S_{\text{pda}}$ 是初始状态。
- ${Z^0}_{\text{下推自动机}} \in \Gamma_{\text{下推自动机}}$ 是初始栈符号。
- $F_{\text{下推自动机}} \subseteq S_{\text{下推自动机}}$ 是接受状态集合。

下推自动机的配置是一个元组 $(s,\ w,\ t)$，其中 $s \in S_{\text{pda}}$ 是当前状态， $w \in \Sigma^*_{\text{pda}}$ 是剩余输入，而$t \in \Gamma^*_{\text{pda}}$ 是当前栈内容。一般（非确定性）下推自动机的转移函数定义为δ_{pda}： $S_{\text{pda}} \times (\Sigma_{\text{pda}} \cup \{\varepsilon\}) \times \Gamma_{\text{pda}} \to \mathcal{P}_{\text{fin}}(S_{\text{pda}} \times \Gamma^*_{\text{pda}})$，它将当前状态、输入符号（或 ε）和栈顶映射为有限个可能的后继状态和栈操作。

还有一种等价的替代定义，其中函数每次只允许压入或弹出一个符号：

$$\delta_{\text{pda}} : S_{\text{pda}} \times (\Sigma_{\text{pda}} \cup \{\varepsilon\}) \times \Gamma_{\text{pda}} \to \mathcal{P}_{\text{fin}}(S_{\text{pda}} \times (\Gamma_{\text{pda}} \cup \{\varepsilon\}))$$

在接下来的证明中，我们将采用后一种限制更多的定义。

下推自动机接受一个输入字 符串的条件是：处理完整个字符串后，满足以下两个条件之一：1. 机器处于 n 接受状态（$s \in$ F_{pda}），或 2. 栈为空。

确定性下推自动机（DPDA）是一种具有以下两项限制的下推自动机：

- 对于任何给定的配置，至多存在一个可能的转移。
- 如果某个状态和栈符号定义了一个 ε-转移，则对于同一状态和栈符号，不得定义其他转移。

我们将通过在 FQDP 与确定性下推自动机之间构造一个双模拟，并展示两个方向上的模拟，来证明这一等价性。FQDP 的递归性和类似栈的特性使其能够由确定性下推自动机建模。

引理 14。 确定性下推自动机（*DPDA*）可以模拟有限任务决策过程（*FQDP*）。

证明。 我们构造一个确定性下推自动机，通过将 FQDP 的组成部分和动作映射到对应的确定性下推自动机元素，来模拟 FQDP。

我们将 FQDP 的组成部分映射到确定性下推自动机的组成部分，如下所示：

- 确定性下推自动机状态：确定性下推自动机的每个状态都代表 FQDP 中一个唯一的局部上下文。我们定义一个双射，$\sigma_{\text{pda}} : \mathcal{V}^{\leq C} \to S_{\text{pda}}$，将所有可能的 FQDP 上下文构成的有限集合映射到确定性下推自动机的状态集 S_{pda}。由于 FQDP 的上下文是有限的，其状态集也是有限的。
- 确定性下推自动机栈：确定性下推自动机的栈模拟 FQDP 的调用-返回层级。当 FQDP 智能体发现一个子任务时，与更新后的局部上下文相对应的状态会被压入栈中。

确定性下推自动机对 FQDP 动作的模拟方式如下：

- 发现输入：确定性下推自动机读取一个输入符号，并转移到一个新状态，该状态反映了更新后的局部上下文以及新的输入节点。
- 发现子任务：确定性下推自动机将更新后的状态（对应带有父标记的局部上下文）压入栈中。随后，它执行一次 ε-转移，进入一个表示新创建的子任务上下文的新状态。
- 完成任务：确定性下推自动机从栈中弹出父状态。随后，它执行一次 ε-转移，进入一个反映父节点更新后上下文的新状态；该上下文现已包含已完成子任务的响应。
- 停止：当确定性下推自动机到达与 FQDP 的终止条件相对应的状态时，它便停止运行。

表 3 展示了这些动作与形式化转移规则之间的详细映射。

在整个模拟过程中，以下不变量始终成立：

- 确定性下推自动机的状态能够准确表示 FQDP 的局部上下文。
- 确定性下推自动机栈顶的符号对应于 FQDP 中父节点的上下文。

由于 FQDP 是确定性的，相应的确定性下推自动机转移可以明确地确定，从而确保这些不变量成立。

因此，这一构造表明，确定性下推自动机能够忠实地模拟 FQDP 的行为。 □

引理 15. 局部上下文大小为 $C = 4$ 的有限任务决策过程（*FQDP*）可以模拟任意确定性下推自动机（*DPDA*）。

FQDP 操作	等价的 DPDA 转移
发现输入 i	$\delta_{\text{pda}}\colon (s, i, Z) \mapsto \left(\sigma_{\text{pda}}(\sigma^{-1}{}_{\text{下推自动机}}(s) + i)\ Z\right)$
发现子任务 q	$\delta_{\text{pda}}\colon (s,\ \varepsilon,\ Z) \mapsto \left(\sigma_{\text{pda}}(q)\,\sigma_{\text{pda}}(\sigma^{-1}{}_{\text{下推自动机}}(s) \leftarrow \pi)Z\right)$
使用 r完成任务	$\delta_{\text{pda}}\colon (s,\ \varepsilon,\ pZ) \mapsto \left(\sigma_{\text{pda}}(\sigma^{-1}{}_{\text{下推自动机}}(p) + \sigma^{-1}{}_{\text{下推自动机}}(s) \leftarrow r)\ Z\right)$
在 $v^{\leq C}$ 停止	在$\sigma_{\text{pda}}(v^{\leq C}) = f$处停机

表 3：FQDP 操作与确定性下推自动机转移规则之间的映射。符号定义如下：σ_{pda} 是从 FQDP 的局部上下文到确定性下推自动机状态的可逆映射。$\sigma^{-1}{}_{\text{pda}}(\ldots)$ 表示从确定性下推自动机状态映射回相应 FQDP 上下文的逆映射。s, $p \in S_{\text{pda}}$ 分别是当前和父确定性下推自动机状态的变量。$f \in F_{\text{pda}}$ 是接受状态的变量。$i,\ q \in \mathcal{V}$ 分别是 FQDP 输入和子任务节点的变量。$r \in \mathcal{R}$ 是焦点节点处响应的变量。$+$ 表示向局部上下文追加新节点。$v \leftarrow r$ 表示以 r 回应节点 v。$v^{\leq C} \in \mathcal{V}^{\leq C}$ 是局部上下文最多包含 C 个节点的 FQDP 局部上下文变量。$Z \in \Gamma_{\text{pda}}$ 是栈顶符号的变量；sZ 表示将状态 s 压入栈。

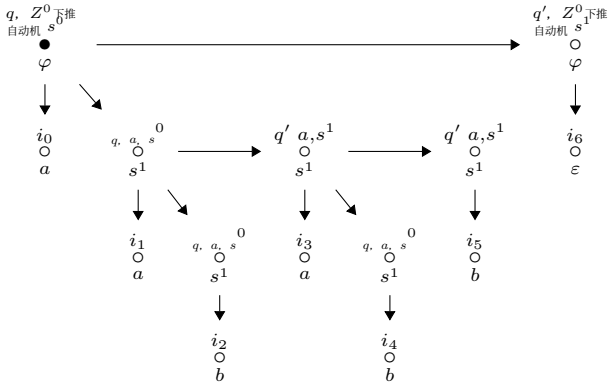


图 6：一个由 $C = 4$ 模拟确定性下推自动机并接受输入字符串 $aababb$ 的 FQDP。输入节点序列（标记为 i_0，i_1、.....）展示了确定性下推自动机消耗输入符号的顺序。φ 是一种特殊的终止响应，表示确定性下推自动机已成功处理输入。箭头仅用于可视化，展示模拟过程中创建的父子关系，并不代表图中的无向边。

最后一个子节点类型	执行
空	读取输入：发现一个新的“输入节点”。如果确定性下推自动机可以进行 ε-转移，新节点的响应可能为空。
输入 a	应用转移：计算 $\delta_{\text{pda}}\colon (s,\ a,\ Z) \mapsto (s',\ \gamma')$。如果 γ' 是压入操作，则发现一个表示 (δ_{pda} ,) 的“子任务”节点，然后用 π 回应，将当前节点标记为父节点并移动焦点到新创建的子任务。否则，如果这是弹出操作，则通过回应完成当前节点并提供新状态 s'，同时将焦点移至父节点。
子任务	创建副本：使用子任务响应中的状态 s 发现一个新的“副本节点”，并从当前焦点节点的目标中复制栈符号。然后，使用 π 回应，以将当前节点标记为父节点，并将焦点移至新创建的副本节点。
副本	退出副本：使用来自副本响应的状态 s回应，并将焦点移至父级任务节点。

表 4：用于模拟确定性下推自动机的代理操作。

证明。我们构造一个 FQDP 来模拟任意确定性下推自动机。为此，我们将确定性下推自动机的配置信息直接编码到 FQDP 的任务节点结构中。

FQDP 任务节点是一个元组 $(g,\ r)$。目标 g 本身也是一个元组 (type $Z,\ s$)，表示该节点在模拟中的角色（type）、来自确定性下推自动机的栈符号（$Z \in \Gamma_{\text{pda}} \cup \{\varepsilon\}$）以及确定性下推自动机状态（$s \in S_{\text{pda}} \cup \{\varepsilon\}$）。

响应 $r \in S_{\text{pda}} \cup \Sigma_{\text{pda}} \cup \{\pi, \varepsilon, \varphi\}$ 用于传递操作结果。这些结果可以是确定性下推自动机状态、输入符号、父标记（π）、空值（ε），也可以是特殊的终止响应 φ，它表示确定性下推自动机已完成对输入的处理，模拟即将终止。

我们定义三种节点类型来驱动模拟：

- i（输入节点），用于消耗输入符号；
- q（任务节点），用于启动子任务以模拟栈压入；以及
- q'（副本节点），即创建的任务节点副本，用于确保智能体的局部上下文大小不超过 C 的限制。

模拟从一个表示确定性下推自动机初始状态且栈为空的任务节点开始。随后，根据焦点节点的子节点配置确定智能体策略，详见表 4，概述如下：

- 无子节点：智能体发现一个新的输入节点，以从确定性下推自动机的输入带中读取下一个符号。
- 最后一个子节点是输入节点：智能体已掌握计算确定性下推自动机转移所需的信息。如果该转移是“压入”，智能体会发现一个新的任务节点，并将其作为子任务处理。如果是“弹出”，智能体会完成当前节点，并将焦点移回其父节点。
- 最后一个子节点是子任务节点：这表示一次“弹出”操作返回。智能体使用子任务响应中的状态以及当前焦点节点目标中的栈符号，发现一个副本节点，然后将该副本作为子任务处理。
- 最后一个子节点是副本节点：这表示智能体正在弹出栈。它会继续沿父节点链向上移动，直到到达一个非副本任务节点。

该过程会持续进行，直到满足确定性下推自动机的接受条件，此时 FQDP 开始执行终止序列。智能体使用特殊终结符 φ 响应当前任务，并将焦点移至其父节点。它重复这一沿父节点链向上移动的过程，直到到达根节点。根节点标记为终结符后，智能体执行“停止”动作，结束模拟。

从确定性下推自动机转移的角度，可以理解该模拟的不变量。不变量状态定义为这样一个时刻：智能体的焦点位于某个任务节点或副本节点上，且该节点最近创建的子节点是一个输入节点。

在此配置中，以下性质成立：

- 智能体掌握了计算下一次确定性下推自动机转移所需的全部信息。
- 确定性下推自动机的当前状态以及栈顶符号存储在焦点节点的目标中。
- 下一个输入符号存储在子输入节点的响应中。
- 焦点节点的最大度始终不超过四，从而确保局部上下文大小保持在 th e 限制 $C = 4$之内。

从该状态开始，智能体策略是确定性的。它计算转移，并执行一系列动作，最终将其带到一个同样满足这一不变量的新状态（或进入停机条件）。
由于智能体的函数仅取决于其局部上下文（由 $C = 4$ 限定），这一构造成功地模拟了任意确定性下推自动机。
□

基于前述两个引理——它们确立了双向的模拟等价——我们得出结论：FQDP 与确定性下推自动机计算等价。

C 非确定性有限任务决策过程的可计算性

定义 16。*A非确定性有限任务决策过程（NFQDP）是 FQDP 的一种变体，由五种主要的分层动作定义：*

- 发现输入：将一个新的输入节点添加为当前焦点节点的子节点。输入节点在被发现时即视为完成。
- 发现子任务：将一个新的、未完成的任务节点添加为当前焦点节点的子节点。
- 追寻子任务：将当前焦点节点标记为父节点，并将焦点转移到其一个未完成的子节点。
- 完成任务：通过在其响应中记录结果，将当前焦点节点标记为已完成，然后将焦点移回其父节点。只有在焦点节点的所有子节点均已完成后，才允许执行此动作。
- 停止：终止计算。只有在任务树的根节点已完成时，才允许执行此动作。

NFQDP 的关键区别在于将“发现”和“追寻子任务”这两种动作分开，使智能体能够沿深度方向遍历预先存在的图结构。

以下推论阐述了这一新模型的计算能力。

推论 17. 非确定性有限任务决策过程（*NFQDP*）与下推自动机（*PDA*）计算等价。

首先，我们证明 PDA 可以模拟 NFQDP，如以下引理所述。

引理 18. 一个下推自 动机（*PDA*）可以模拟非确定性有限任务决策过程（*NFQDP*).

证明。 我们在附录 B 中证明的基础上，进一步处理 NFQDP 中发现子任务与追寻子任务这两个动作的分离。大多数动作的模拟方式保持不变，主要变化如下：

- 发现子任务：下推自动机执行一次 ε-转移，进入一个新状态，该状态反映 NFQDP 更新后的局部上下文，其中包含新任务节点。
- 追寻子任务：下推自动机将当前状态（表示带有父标记的父节点上下文）压入栈中。随后，它执行一次 ε-转移，进入一个表示子任务上下文的新状态。此步骤利用下推自动机的非确定性，选择与预构建树结构中现有局部上下文相匹配的正确转移。
- 其他动作（发现输入、完成任务、停止）的模拟方式与 FQDP-DPDA 模拟中的对应动作类似。

下推自动机的转移函数可以利用其固有的非确定性来模拟 NFQDP。在智能体追寻子任务时，这对于确定其状态是必要的，因为目标节点的局部上下文可能因预构建图结构而异。根据定义，NFQDP 中可能的局部上下文集合是有限的，因此可以构造有限组下推自动机转移规则，以覆盖所有可能的配置。 □

对于证明的另一个方向，使用 NFQDP 直接模拟 PDA 会因 PDA 的非确定性 ε 转移而变得复杂。因此，我们将改为证明：可以构造一个 NFQDP，使其接受任意给定上下文无关语言（CFL）中的任意字符串；而上下文无关语言正是 PDA 所识别的语言类（Hopcroft 等人 [2001]）。

上下文 无关文法（CFG）是由四元组 $(N_{\text{cfg}},\ \Sigma_{\text{cfg}},\ P_{\text{cfg}},\ S_{\text{cfg}})$ 定义的形式文法，其中 e:

- N_{CFG} 是一个有限的非终结符集合。
- Σ_{CFG} 是一个有限的终结符集合。
- $S_{\text{CFG}} \in N_{\text{CFG}}$ 是开始符号。
- P_{CFG} 是由形式为 $A \to \gamma$ 的产生式规则组成的有限集合，其中 $A \in N_{\text{CFG}}$ 是一个非终结符，而 $\gamma \in (N_{\text{CFG}} \cup \Sigma_{\text{CFG}})^*$ 是由终结符和非终结符组成的字符串。

上下文无关语言（CFL）是由上下文无关文法生成的语言。

每个上下文无关文法都可以转换为等价的***Chomsky*** 标准形 （CNF）（Chomsky[1959]）文法，其中所有产生式规则都必须采用以下形式之一：

- $A \to BC$，其中 $A,\ B,\ C$ 是非终结符。

- $A \to a$，其中 A是非终结符， a 是终结符。
- $S_{\text{CFG}} \to \varepsilon$，其中 S_{CFG}是开始符号。只有当该语言包含空字符串时，才允许使用此规则；如果存在此规则，开始符号就不能出现在任何其他规则的右部。

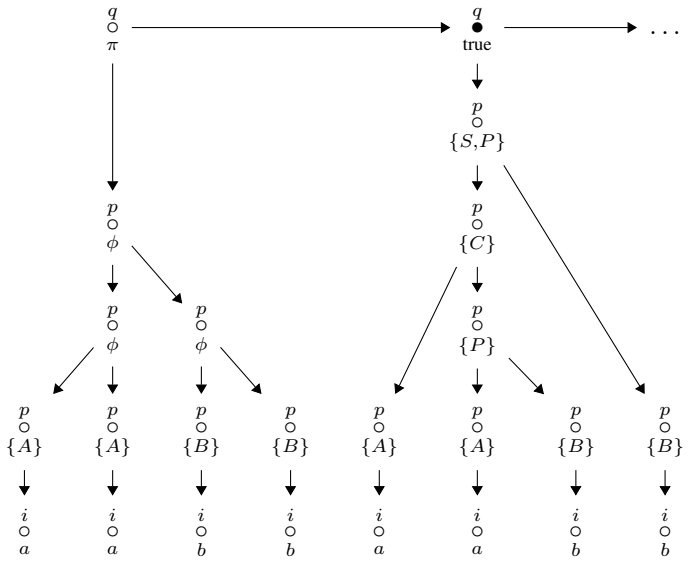


图 7：对输入字符串 $aabb$进行 NFQDP 模拟时的解析图。该图表示根据以下乔姆斯基范式 CFG 得出的所有可能解析树的空间： $S \to CB \mid AB$、 $P \to CB \mid AB$、$C \to AP$、 $A \to a$ 和 $B \to b$。

引理19。 可以构造一个局部上下文大小为 $C = 3$ 的非确定性有限任务决策过程（*NFQDP*），以接受任意给定的上下文无关语言（*CFL*）。

证明。 给定一个输入字符串和一个乔姆斯基范式文法，我们首先预先构建一个图，用来表示该字符串所有可能解析树的空间。由于乔姆斯基范式解析树是二叉树，即任何节点最多有两个子节点和一个父节点，因此使用局部上下文大小为 $C = 3$ 即可完成这一构建。随后，智能体的任务是对这个预先构建的图进行深度优先遍历，以找到一棵符合该文法产生式规则的解析树。

该图的结构是一串以根节点相连的二叉树，每棵二叉树都表示输入字符串的一种可能解析树。我们使用三种节点类型来构造此解析图：

- i （输入节点）：输入节点是解析树中的叶节点。其响应包含输入字符串中对应的终结符。
- q （任务节点）：任务节点是解析树的根节点。其响应是一个布尔值，表示该解析或后续任意解析是否有效。
- p （产生式节点）：产生式节点表示一条文法规则的应用。其响应包含能够生成其子树所覆盖子串的所有非终结符集合。

所有节点的目标字段仅指定该节点属于集合 $\{i, q, p\}$ 中的哪种节点类型。图 7 展示了针对输入字符串 $aabb$ 的这一构造。

智能体从根节点出发，沿深度方向遍历该图，通过执行子任务来探索解析树的不同分支。在每个产生式节点处，智能体会检查文法，找出右部由该节点子节点对应的非终结符构成的规则。随后，它收集所有匹配规则的左部非终结符，并将其存储在响应字段中。

返回解析树的根节点后，智能体会沿链推进到下一个任务节点，以评估下一个可能的解析树。这一前向遍历会持续进行，直到智能体探索完图中所有可能的解析树。

此时，终止序列开始沿任务节点链向后移动。对于每个任务节点，智能体都会在其响应字段中记录一个布尔值。如果该节点自身的解析树有效（即其根节点包含文法的开始符号），或者后续任务节点的响应已经是“true”，则响应设为“true”。否则，响应为“false”。完成这次反向遍历后，智能体会到达整个图的初始根节点并停止。

由于文法处于乔姆斯基范式，输入字符串的任何解析树都是一棵二叉树。这种结构确保可能的解析树总数有界，且等于与输入字符串长度相对应的卡特兰数。此外，由于产生式规则集合是有限的，任何产生式节点的响应中可能出现的所有非终结符组成的集合也是有限的。这两个条件共同保证预构建图的大小有界，并确保对于任何非空输入字符串，遍历过程都会终止。

最后，空字符串这一特殊情况会直接处理。如果语言包含空字符串，则可以将智能体设计为在输入预构建时立即停止并接受，预构建内容为单个任务节点。这一构造表明，可以构建一个 NFQDP 来接受任意给定的上下文无关语言。

□

D 参考增强型有限任务决策过程的可计算性

定义 20. 一种参考增强型有限任务决策过程（*RQDP*）是在 *FQDP* 的基础上增加了三个用于实现记忆检索的组成部分：

- 一个非有限参考集合 $\mathcal{T}$，其元素用作节点的标签。多个节点可以共享同一引用。该集合配备了相等运算符 $\mathcal{T} \times \mathcal{T} \to \{\textit{true}, \textit{false}\}$。
- 一个引用生成函数，τ： $\mathcal{T} \times \mathcal{G} \to \mathcal{T}$。每当创建新节点时，都会将 τ 应用于父节点的引用和新节点的目标，以计算其引用。
- 一个检索动作，$\mathcal{V}^{\leq C} \to \mathcal{G}$。智能体执行此动作时，会输出一个目标。随后，*Quest Graph* 会创建一个新节点，将其连接到焦点节点，并使用 τ 函数为其分配引用。然后，它会使用此新引用，在历史记录中检索与匹配引用相关联的最新响应；接着将该响应分配给新节点。如果不存在匹配项，则将新节点的响应设为空符号 ε。

一种非确定性参考增强型有限任务决策过程*(NRQDP)*是 *RQDP* 的一种变体，它结合了 *NFQDP* 的非确定性分层动作。

现在我们将证明 RQDP 具备图灵完备性。具体而言，我们将证明它能够模拟使用单向无限纸带的标准图灵机（Hopcroft 等人 [2001]）。由于非确定性 NRQDP 能够模拟其确定性对应形式，因此可知 NRQDP 同样具备图灵完备性。

定理 21。 在参考增强型有限任务决策过程（*RQDP*）下运行、具有 $C = 4$ 本地上下文容量的智能体能够模拟任意图灵机。

证明。 为了模拟图灵机，我们首先定义构造中所使用的 RQDP 节点结构。每个节点的目标是一个元组（类型 s、t），表示节点的类型、图灵机状态 $s \in S_{\text{tm}}$ 和纸带符号 $t \in \Gamma_{\text{tm}}$。响应是一个元组（s'、t'），表示下一个图灵机状态以及要写入的符号。我们定义四种节点类型来驱动模拟：

- q (任务节点)：用于启动子任务，模拟磁头向右移动。
- q' (副本节点)：任务节点的副本，用于管理局部上下文大小，作用与 FQDP-DPDA 模拟中的作用类似。
- f (获取节点)：用于获取纸带符号。
- w (写入节点)：用于记录纸带符号。

模拟策略利用这一机制以两种不同的方式表示图灵机的纸带：磁头左侧的部分由 FQDP 类似栈的层次结构维护，而右侧的部分则由引用与检索系统管理。

我们将参考集合定义为纸带位置的集合 $\mathbb{Z}^+$，其中引用 $\xi \in \mathbb{Z}^+$ 表示当前纸带位置。引用生成函数 τ 定义为模拟磁头向右移动。

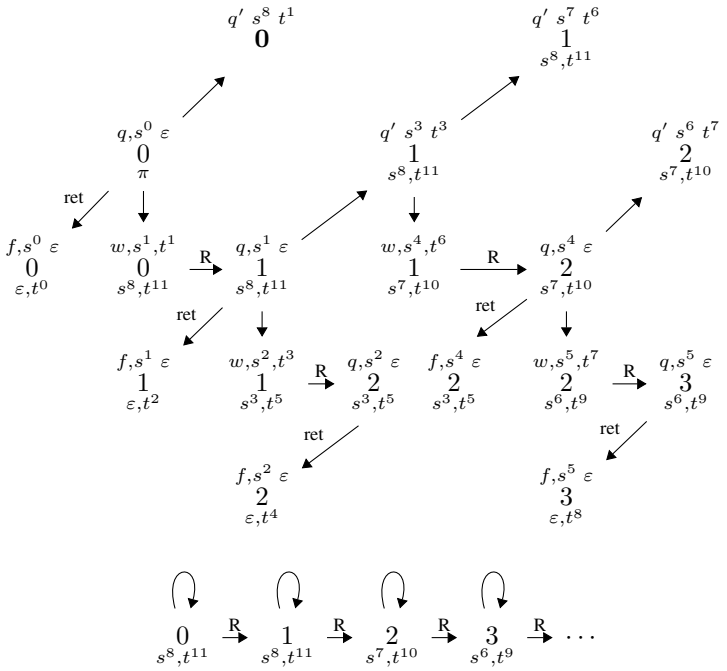


图8：一个以 $C = 4$模拟图灵机的 RQDP。上方：Quest Graph 执行轨迹，其中父子箭头仅用于可视化，并不表示图的无向边。标记为“ret”的边表示检索操作，而“R”对应图灵机读写头向右移动。图中模拟遵循移动序列$RRLRRLLL$，从标有父节点符号（π）的任务节点开始，到带有粗体引用的节点结束。下方：对应的引用图，其中每个编号节点都存储该引用的缓存响应。在图示中，状态序列表示为 s^0、s^1，.....，纸带符号按创建顺序表示为 t^0、t^1，.....。对角布局是一种风格选择，旨在直观地使向右移动与页面上的向右方向一致，并确保检索操作箭头清晰地指向最近更新的节点。

当智能体创建一个新的任务节点（类型 q）时，该函数将引用递增至下一个纸带位置：τ：$(\xi,\ (q,\ \cdot,\ \cdot)) \mapsto \xi + 1$。$\cdot$符号是通配符，表示无论目标中的状态或纸带符号为何，该规则都适用。对于所有其他节点类型或条件，引用保持不变。

智能体策略是一组基于焦点节点类型及其子节点状态的确定性规则。模拟周期可按如下方式构建：

1. 当焦点位于任务节点上时：智能体的首要目标是收集计算图灵机转移所需的信息。

   (a) 如果节点没有子节点，智能体首先需要读取当前纸带符号。它执行检索操作以创建一个获取节点。根据引用生成函数，该新的获取节点被赋予与其父任务节点相同的引用。这确保系统从具有该引用的节点的最新响应中检索符号，从而正确获取当前读写头位置上的纸带符号。(b) 如果节点有一个获取节点作为子节点，智能体现在同时拥有状态 s（来自其自身的目标）和纸带符号 t（来自获取节点的响应）。它计算图灵机转移 δ_{tm}：$(s,\ t) \mapsto (s',\ t',\ d')$。后续动作取决于移动方向 $d' \in \{L, R\}$：

   - 如果 $d' = R$（向右移动），智能体会发现一个写入节点来存储新符号。目标设为（$w,\ s',\ t'$）后，它会将焦点移至这个新的写入节点。
   - 如果 $d' = L$（向左移动），智能体会以（$s',\ t'$）回应当前节点，从而完成该节点。随后，它将焦点移至其父节点。

（c）如果该节点有一个写入节点作为子节点，则表示智能体正在执行向左移动。它会发现一个新的副本节点，并以（q'， s， t）作为目标，其中 s 来自写入节点的响应， t 来自写入节点的目标。随后，它将该副本节点作为子任务来执行。（d）如果该节点同时有一个写入节点和一个副本节点作为子节点，则表示智能体正从一个副本节点返回（向左移动）。智能体会以已完成副本节点的结果进行回应，并将焦点移至其父节点，继续向左移动。

2. 当焦点位于副本节点时，大多数情况与任务节点类似：

(a) 如果该节点没有子节点，智能体会使用其目标中的状态和纸带符号来计算图灵机转移，然后执行条目 1b 中描述的相同动作。(b) 其余情况与条目 1c 和 1d 中描述的情况相同。

3. 当焦点位于写入节点时：智能体的目标是启动模拟的下一个向右步骤。

(a) 如果该节点没有子节点，它会发现一个目标为 (q， s， ε) 的任务节点，其中 s 来自写入节点的目标。然后，它将焦点移至这个新任务节点，以开始下一个模拟周期。(b) 如果该节点已经有一个任务节点子节点，这表示计算中更深层的部分（向左移动）已经返回。智能体会用已完成子节点的结果进行回应，并将焦点移至其父节点，继续向左移动。

这一循环持续进行，直到图灵机到达停机状态，此时 RQDP 使用与 FQDP 相同的终止过程来停止计算。图8展示了一个使用 $C = 4$ 模拟图灵机的 RQDP。
初始纸带配置在设置阶段建立。该过程从一个位于引用$\xi = 0$处的任务节点开始。从这一初始位置出发，一个特殊的初始化智能体将输入字符串填入图中。它通过反复执行上述模拟过程中的向右移动序列来完成这一操作，但有一处修改：用发现输入动作替代检索操作，将纸带符号直接写入新创建的节点。初始化完成后，主模拟智能体接手，其焦点从链中的最后一个任务节点开始，该节点对应于初始纸带最右端。

该模拟的不变量根据图灵机的转移来定义。不变量状态是指智能体的焦点位于副本节点，或位于一个任务节点上的时刻；在后一种情况下，该任务节点最近创建的子节点是一个获取节点。

在此配置下，以下性质成立：

- 智能体拥有计算下一次图灵机转移所需的全部信息。
- 图灵机的当前状态存储在焦点节点的目标中。
- 根据焦点节点的类型获取当前读写头位置上的纸带符号：如果焦点是任务节点，则该符号位于其获取子节点的响应中；如果焦点是副本节点，则该符号位于副本节点自身的目标中。
- 焦点节点的最大度不超过四，从而确保局部上下文大小保持在 $C = 4$的限制范围内。

从此状态出发，智能体的策略是确定性的。它计算转移并执行一系列动作，最终到达一个同样满足此不变量的新状态，或达到停机条件。
该构造表明，具有 $C = 4$ 的 RQDP 可以模拟图灵机。 □

E 语言模型的可计算性

设语言模型定义为一个元组 $LM = (\Gamma_{\text{lm}}, \delta_{\text{lm}}, C)$，其中：

- Γ_{lm} 是一个有限的词元集合（即词汇表）。
- δ_{lm}： $\Gamma^{C}{}_{\text{lm}} \to \Gamma_{\text{lm}}$ 是转移函数，它将一个由 C 词元组成的序列映射为下一个词元。
- $C \in \mathbb{Z}^{+}$ 是上下文长度，且为有限值。

有限状态机（FSM）可以定义为一个元组（$S_{\text{fsm}}, \Sigma_{\text{fsm}}, \delta_{\text{fsm}}, s^{0}_{\text{fsm}}, F_{\text{fsm}}$），其中：

- $S_{\text{有限状态机}}$是一个有限状态集。
- Σ_{fsm} 是一个由输入符号构成的有限集合（输入字母表）。
- δ_{fsm}： $S_{\text{fsm}} \times \Sigma_{\text{fsm}} \to S_{\text{fsm}}$ 是转移函数。
- $s^0{}_{\text{fsm}} \in S_{\text{fsm}}$ 是初始状态。
- $F_{\text{fsm}} \subseteq S_{\text{fsm}}$ 是接受状态集合。

引理 22。 上下文大小为 $C = 2$ 的语言模型（*LM*）可以模拟有限状态机（*FSM*）。

在实际应用中，语言模型的上下文会通过追加用户输入来更新。为了进行这种模拟，我们采用相同的机制，将有限状态机的输入符号输入到语言模型的上下文中。

证明。 给定一个有限状态机，我们可以按如下方式构造一个上下文大小为 $C = 2$ 的语言模型。首先，我们将语言模型的词汇表定义为有限状态机的状态集与输入字母表的并集： $\Gamma_{\text{lm}} = S_{\text{fsm}} \cup \Sigma_{\text{fsm}}$。其次，我们定义语言模型的转移函数 δ_{lm}，使其完美模拟有限状态机的转移函数；具体而言，对于任意状态 $s \in S_{\text{fsm}}$ 和输入符号 $a \in \Sigma_{\text{fsm}}$，令 $\delta_{\text{lm}}(s,\ a) = \delta_{\text{fsm}}(s,\ a)$。

模拟从上下文中的有限状态机初始状态 s^0_{fsm} 开始。对于输入字符串中的每个符号，该过程交替执行以下两个步骤：

- 首先，将当前输入符号追加到上下文中。
- 其次，语言模型将其转移函数应用于上下文（此时上下文包含当前状态和输入符号），以生成并追加下一个状态。

输入字符串 i 若处理完整个字符串后，上下文中的最终状态属于 F_{fsm}，则该字符串被视为已接受 .
这一构造表明，具有 $C = 2$ 的语言模型可以模拟有限状态机。 □

引理 23. 存在一个有限状态机（*FSM*），可以模拟语言模型（*LM*).

证明。 令 $\sigma_{\text{lm}} : \Gamma_{\text{lm}}^{C} \to S_{\text{fsm}}$ 为从所有可能的 LM 上下文集合到 FSM 状态集 S_{fsm} 的双射。由于 LM 的词汇表（Γ_{lm}）和上下文长度（C）都是有限的，状态集 S_{fsm} 也必定是有限的。对于此构造，我们将 FSM 的输入字母表设为 LM 的词汇表 $\Sigma_{\text{fsm}} = \Gamma_{\text{lm}}$。

我们定义 FSM 的转移函数 δ_{fsm}，以模拟 LM 的两个主要动作：

- 自回归生成：当 LM 从其当前上下文（t_1，……，t_C）内部生成一个词元，得到 t_{C+1} 时，FSM 会从状态 $\sigma_{\text{lm}}(t_1$，……，$t_C)$ 执行一次 ε 转移，到达新状态 $\sigma_{\text{lm}}(t_2$，……，t_C，$t_{C+1})$。
- 消耗外部输入：当外部输入词元 a 被追加到 LM 的上下文中时，FSM 会根据该符号执行转移。从状态 $\sigma_{\text{lm}}(t_1$，……，$t_C)$ 出发，FSM 读取输入 a，并转移到新状态 $\sigma_{\text{lm}}(t_2$，……，t_C，$a)$。

有限状态机的初始状态由语言模型的初始上下文定义。通过这些转移，有限状态机在任意时刻的状态都能完美跟踪语言模型的上下文。这一构造表明，有限状态机可以模拟有限上下文语言模型的行为。 □

结合前两个引理，我们可以得出
定理 24. 具有有限上下文的语言模型与有限状态机（*FSM*）计算等价。

F 有界最大依赖计算图分析

最大依赖计算图（MCG）是一种有向无环图，其节点具有唯一的顺序，并且每个节点都接收来自所有顺序较低节点的入边。
引理25. 对于任意计算图，都存在一个与之等价的最大依赖计算图（*MCG*），它保留所有原有依赖项，且最多只需添加一个节点即可构造。

证明。该结论 构建过程包括为节点创建全序，然后添加边以形成 MCG .

1. 首先，我们通过找出所有终端节点（没有出边的节点）来确定顺序。如果存在多个终端节点，则创建一个新的单一终端节点，将其连接到所有原有终端节点，并将其顺序设为 0。

2. 随后，我们反向遍历该图：将顺序为 k 的节点的直接前驱节点赋予顺序 $k+1$，并重复此过程，直到所有节点都被排序。如果某个节点已经排序，则取其现有顺序与新顺序中的最大值。由于该图是有向无环图，此过程必定会终止。随后，为所有顺序相同的节点任意指定子顺序，从而形成一个全序。

3. 最后，对于每一对节点，如果一个节点在全序中先于另一个节点，且两者之间尚不存在边，则添加一条边。

此过程会生成一个 MCG，其中每个节点都依赖于所有先前的节点。关键的是，它保留了原图的优先关系，因为我们的反向遍历排序确保：如果原图中存在一条边，那么源节点在最终排序中必定先于目标节点。 □

有界最大依赖计算图（BMCG）是 MCG 的一种扩展形式，具有以下属性：

- 存在一个常数 $C \in \mathbb{Z}^+$，用于定义任何节点允许的最大入度（依赖项数量）。
- 原始 MCG 中入度大于 C 的任何节点都会转换为由代理节点构成的分层树。该结构将原始节点连接到其所有依赖项，同时确保生成的图中任何节点的入度均不超过 C。

BMCG 中新增代理节点和边的总数是两个参数的函数：原始 MCG 中的节点总数（N）和最大入度约束（C）。以下是对这种关系的详细分析：

设 $k(d)$ 表示入度为 d 的节点所需的代理节点数量（见表5）。

$$k(d) = \begin{cases} \left\lfloor \frac{d-2}{C-1} \right\rfloor, & \text{if } d \geq 2 \\ 0, & \text{otherwise} \end{cases}$$

设 $K = k(N-1)$ 表示入度可能达到最大值 $N-1$ 的节点所需的代理节点数量。

$$\begin{aligned} K &= \left\lfloor \frac{N-3}{C-1} \right\rfloor \\ &= O(N) \end{aligned}$$

设 N^+ 为 BMCG 中新增代理节点的总数。

$$\begin{aligned} N^+ &= \sum_{d=0}^{N-1} k(d) \\ &= (C-1)\left(\frac{K(K+1)}{2}\right) - K\left[(C-1)K - [(N-1)-C]\right] \\ &= K(N-2) - (C-1)\left(\frac{K(K+1)}{2}\right) \\ &= O(N^2) \end{aligned}$$

因此，可以确定所得 BMCG 的最终规模。由于原始 MCG 包含 N 个节点和 $O(N^2)$ 条边，且新增的代理节点（N^+）数量同样为 $O(N^2)$，最终图由 $N + N^+ = O(N^2)$ 个节点和 ${}^{N}{}_{(N-1)_2} + N^+ = O(N^2)$ 条边组成。

C	$d=1$	2	3	4	5	6	7	8	9	10	11	...
2	0	0	1	2	3	4	5	6	7	8	9	
3	0	0	0	1	1	2	2	3	3	4	4	$\left\lfloor \frac{d-2}{C-1} \right\rfloor$
4	0	0	0	0	1	1	1	2	2	2	3	

表5：对于单个 d 度节点，在不同上下文大小约束（C）和原始节点入度（d）下所需的代理节点数量。