arXiv:2302.13482v3 [cs.LO] 4 Mar 2023

PyReason：开放世界时序逻辑软件

Dyuman Aditya[†]，Kaustuv Mukherji[*,†]，Srikar Balasubramanian，AbhirajChaudhary 和 PauloShakarian[*]

亚利桑那州立大学，美国亚利桑那州坦佩市南米尔大道 *699* 号，邮编 *85281*

摘要

神经符号推理日益普及，推动了各种形式的可微（即模糊）一阶逻辑的应用。我们介绍 PyReason，这是一种基于广义注释逻辑的软件框架，既涵盖当前各类可微逻辑，也纳入时序扩展，以支持在有限时间段内进行推理，并具备开放世界推理能力。此外，PyReason 的实现可直接支持对图结构（例如知识图谱、社交网络、生物网络等）进行推理，生成完全可解释的推理轨迹，并包含类型检查和内存高效实现等多种实用功能。本文回顾了集成到我们实现中的广义注释逻辑的各种扩展、我们现代高效的 Python 实现（能够执行精确且可扩展的演绎推理），以及一系列实验。PyReason 可通过以下网址获取：**github.com/lab-v2/pyreason**。

关键词

逻辑程序设计，神经符号推理，广义注释逻辑，时态逻辑，一阶逻辑，开放世界推理，图推理，人工智能工具

1. 引言

各种神经符号框架都利用底层逻辑来支持模糊逻辑 [1], 参数化 [2], 和可微结构 [3]等功能。通常，这类框架的实现会针对所用的特定逻辑编写定制的演绎软件，这限制了模块化和可扩展性。此外，新兴的神经符号应用场景，包括有限时间段上的时态逻辑 [4] 和知识图谱推理 [5]，都需要一个涵盖广泛功能的逻辑框架。幸运的是，广义注释逻辑 [6] 及其各种扩展 [7, 8, 9] 涵盖了其中许多功能。**本文介绍一个名为 PyReason 的新软件包，它使用广义注释逻辑进行演绎，涵盖了各种神经符号框架所需的许多功能，包括模糊、开放世界、时态和基于图的推理**。具体而言，PyReason 的核心功能是对以下内容进行推理

载于 *A. Martin*、*K. Hinkelmann*、*H.-G.Fill*、*A.Gerber*、*D. Lenat*、*R. Stolle*、*F. van Harmelen*（编），《*AAAI 2023* 春季研讨会论文集：机器学习与知识工程结合所面临的挑战》（*AAAI-MAKE 2023*），凯悦酒店，旧金山机场，加利福尼亚州，美国，*2023*年*3*月*27*日至*29*日。*通讯作者。

[†]这些作者贡献相同。 kmukherji@asu.edu (K. Mukherji); pshak02@asu.edu (P. Shakarian) https://search.asu.edu/profile/4179815 (K. Mukherji); https://labs.engineering.asu.edu/labv2/ (P. Shakarian) 0000-0002-4889-3499 (D. Aditya); 0000-0001-8044-1110 (K. Mukherji)©2023年本文作者版权所有。允许按照知识共享署名 协议（CC BY 4.0）使用。CEUR研讨会论文集（CEUR-WS.org)

一阶逻辑（FOL）和命题逻辑语句，这些语句可以用格结构中的元素或该格上的函数进行注释。此外，我们还提供了额外的实用语法和语义扩展，使其能够对知识图谱、时态逻辑、各种网络扩散模型以及谓词-常量类型检查约束进行推理。此实现提供了快速且经过内存优化的不动点算子实现，该算子用于演绎过程。通过直接实现不动点算子（而非采用黑箱启发式方法），该软件能够充分解释结果。因此，该框架不仅涵盖经典逻辑，也涵盖了多种其他逻辑框架，包括模糊逻辑 [10, 11, 12], 逻辑神经网络中使用的加权实值逻辑 [2], van Emden 逻辑 [13], Fitting 双格逻辑 [14], 各种用于图或社交网络推理的逻辑框架 [9, 8, 15] （以及这些框架所涵盖的各种网络扩散模型），或许最重要的是，还包括可通过可微归纳逻辑编程学习语法结构的逻辑框架 [3, 16] 以及其他神经符号框架 [17, 7]。我们方法的主要优势如下：

1. **直接支持知识图谱上的推理**。 知识图谱结构是符号数据最常用的表示形式之一。虽然像 [18] 这样的黑箱框架也支持在图结构上进行推理，但它们无法提供我们的方法所具备的可解释性。2. **支持注释**。 Prolog [19] 和 Epilog [20] 等经典逻辑实现本质上不支持注释或注释函数，因此也无法直接支持模糊算子等功能。此外，我们的框架不仅支持模糊算子，还允许使用任意函数处理实值或实数区间。这是推理神经符号方法（如 [2, 3, 16, 17, 7]）所学习的构造时的一项关键优势。3. **时序扩展**。 尽管已有研究表明， [6] 的框架能够捕捉各种时序逻辑，但 [9] 等扩展提供了句法和语义上的附加机制，可显式表示时间，并支持在有限时态序列上进行时序推理。遵循 [9], , 我们采用表示多个时间点的语义结构，但通过紧凑的方式实现，以节省内存。我们的解决方案支持规则的模糊版本，例如“如果 $q(A)$，那么在 t 个时间步内 $r(A)$。”请注意，目前几乎所有模糊逻辑实现都不具备这些功能。4. **使用解释**。 我们将解释定义为同时作用于谓词和时间的带注释函数。它使我们能够捕捉在 $t = 0$之前为真的事实。尽管带注释逻辑 [6] 无需额外构造即可涵盖各种时序逻辑，但我们通过在解释中纳入时态成分来实现时序推理。通过结合带注释谓词和时间变量，我们认为我们的框架更加灵活，也更适用于新兴的、涉及时间的神经符号应用，因为这类应用本质上既需要时间，也需要实值注释。此外，需要指出的是，我们不采用封闭世界假设，即初始解释集合中未提及的内容都是 $false$。相反，我们认为在时间开始时，所有其他解释的状态都是未知的。5. **图形知识结构**。 我们还实现了 [8] ， 它提供图形

为 [6]提供句法扩展。这已纳入我们的实现，尤其是添加了用于在此类结构中进行推理的扩展句法算子（例如，要求存在 k 个项目的存在算子）。这类规则的一个例子是模糊版本的“如果 $q(A)$，并且存在 k 个 B，使得 $b(A, B)$，那么 $r(A)$”。[1]

6. **通过基元化降低计算复杂性**。 我们的软件利用图结构固有的稀疏性，并采用一种新颖的谓词-常量类型检查约束实现，这不仅显著提升了软件在各种应用领域中的实用性，也大幅降低了基元化问题引起的复杂性。据我们所知，还没有其他一阶逻辑框架同时具备这两项能力。

7. **能够检测并解决推理过程中的不一致性**。 由于逻辑推论是通过对预定义逻辑规则应用不动点算子推导得出的，因此不仅能够检测逻辑不一致，还能精确定位不一致发生在推理过程中的何处。我们通过利用不确定性来解决此类不一致。在软件实现中，一旦检测到不一致，我们就会放宽边界并将其修正为完全不确定。检查并定位不一致的能力增强了可解释性。诸如 [2, 7] 这样的神经符号方法也可能会考虑在训练阶段将不一致性纳入损失。

在第 2 节中，我们概述了 [6] 的语法和语义，以及我们的扩展。我们的软件实现将在第 3 节中介绍，并在仅在线补充材料中进一步阐述。在第 4 节中，我们提供了框架的实验结果，以展示其在两个不同现实领域中的推理能力。我们在一个供应链 [21]（$10K$ 常量）和一个社交媒体 [22]（$1.6M$ 常量）数据集上进行了实验。为进行评估，我们使用了各种经过人工整理的逻辑程序，这些程序规定了图的时间演化、图的补全以及其他实际应用场景（例如识别潜在的供应链中断）的规则，并研究了各个因素如何影响运行时间和内存使用量（例如常量数量、谓词数量、时间步、推理步骤等）。结果表明，在较大的范围内，运行时间和内存几乎保持不变，随后随着网络规模的增大呈次线性增长。

在线资源

开源 Python 库可访问： **pypi.org/project/pyreason**。PyReason 代码库可访问：**github.com/lab-v2/pyreason**。仅在线补充材料可访问：**github.com/lab-v2/pyreason/tree/main/lit**

2. 逻辑框架

本节将概述带注释逻辑框架，并从高层次介绍逻辑构造、知识图谱结构、关键优化，以及

[1]请注意，尽管此示例是经典逻辑，但 PyReason 支持完全带注释逻辑，允许任意定义模糊算子（例如，t-范数）；技术细节请参见2 及在线补充材料。

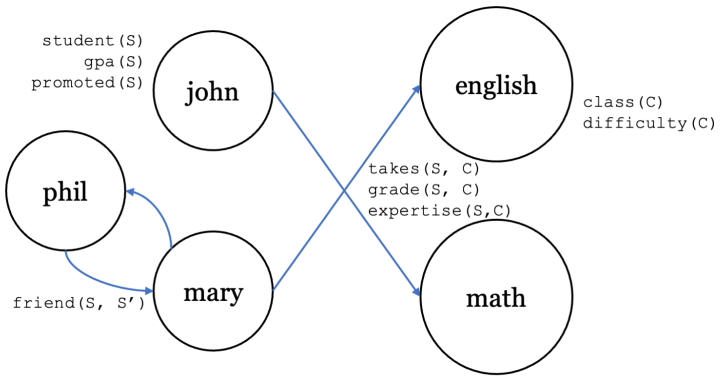

图1： 知识图谱示例

不动点算法的运行方式。

知识图谱。我们假设存在一种图结构 $G = (\mathcal{C}, E)$，其中节点也是一阶逻辑框架中的常量（记作集合 $\mathcal{C}$）。边记作$E \subseteq \mathcal{C} \times \mathcal{C}$，用于指定任意类型的关系是否可以存在于两个常量之间。与近期结合知识图谱和逻辑的框架类似 [3, 18], ，我们假设语言中的所有谓词要么是一元谓词（可视为标记节点），要么是二元谓词（可视为标记边）。我们指出，假设存在一个特殊的二元谓词 rel，并将其视为保留字。对于 $(a, b) \in E$，我们将 rel (a, b) 视为重言式；对于 $(a, b) \notin E$，我们将 rel (a, b) 视为不确定。请注意，通过创建 G 作为全连接图，我们可以支持常量配对之间不受任何限制的情况。同样，我们只需使用一个单节点图，就能轻松支持命题情形（本质上是将一元谓词视为地面原子）。本节将贯穿一个示例。在图1中，我们展示了如何在我们的框架中指定知识图谱。

示例2.1（知识图谱）。考虑以下节点：三名学生——*Phil*、*John*、*Mary*，以及两门课程——英语和数学。节点和边具有一元和二元谓词，如图所示。*1*。因此，我们得到以下非基原子：*student(S), gpa(S), promoted(S)class(C), difficulty(C)friend(S,S')takes(S,C), grade(S,C), expertise(S,C)*这里，*S*、*S'* 和 *C* 是变量，用图中的常量对它们进行实例化后，就会生成如下基原子：*student(john), student(phil),student(mary)class(math), class(english)takes(john,math), takes(mary,english)...*

在命题情形下，非基原子会简化为命题陈述。例如，谓词*“takes(john,math)”*可以表示为命题陈述：*“John* 修读数学课*”*，其值可以为真或假。本例中它为真，如图所示。*1*。

实值区间标注。 带注释逻辑 [6] 的一项关键优势，是能够使用格结构中的元素以及定义在该格上的函数来标注框架中的原子。在我们的软件中，我们使用由区间组成的下格结构，这些区间是 $[0,1]$ 的子集。这与模糊算子 [12], 的真值区间，以及神经符号推理 [2, 7], 和社交网络分析 [8, 9]中的范式直接一致。我们只需将操作限制在区间的下界，并将上界保持为 1，即可完全支持标量值标注。这些注释可以通过将其限制为 $[0,0]$（假）和 $[1,1]$（真）来支持经典逻辑。通过允许 $[0,1]$, （表示未知），它也可以支持三值逻辑。当然，没有必要施加这些限制，尤其是在希望支持充分利用整个区间[2, 8, 9]的逻辑时。此外，我们还支持 [7]中详细介绍的文字。我们以与 [1] 相同的方式处理否定——对于一个以 $[\ell, u]$ 标注的原子，我们使用 $[1-u, 1-\ell]$ 来标注其强否定（¬）。

示例 2.2（实值区间标注）。继续前面的示例，我们可以支持上述各种注释。命题逻辑：*student(john):*`[1,1]`（真命题示例）*takes(mary,math):*`[0,0]`（假命题示例）模糊逻辑（使用标量值）：*gpa(john):*`[X,1]`，$X \in$ `[0,1]`完整区间使用：*difficulty(english):*`[0.3,0.7]`（这里使用了两个边界，以反映学生对该学科*“*英语*”*难度的感知差异）。不确定性建模和／或三值逻辑：假设我们并未完全了解这个网络——具体来说，我们没有任何关于 *John* 和 *Phil* 之间友谊的信息。因此，他们可能是朋友（注释为*[1,1]*），也可能不是朋友（注释为 *[0,0]*）。我们的框架可以将这种情况建模为：*friend(john,phil):*`[0,1]`

解释。 在逻辑框架中，通常会使用一组初始事实。我们使用*“*初始解释*”*一词来指代程序开始时成立的注释。在设想的领域中——包括我们进行实验的领域——这些初始解释应表示为知识图谱，其中不仅包含图G，还包含节点和边上的属性（类似于谓词）以及实值区间标注（用于指定每个元素的初始注释）。此外，遵循各种同时纳入时间标注和其他实值标注 [9, 8, 23, 24, 25], 的时序逻辑框架所体现的思路，我们扩展语法，将时间标注纳入解释的一部分。根据相关研究，时间表示为有限的离散时间点。初始解释包含了在时间 0 之前应视为真的内容。此外，通过初始解释，我们可以指定谓词是静态的（换言之，由这些谓词构成的地面原子在所有时间段内保持相同的注释），还是非静态的（允许发生变化）。

）。在某些领域中，添加这一限制显然具有优势，同时也能提升跨时间段推理的关键实现效率。此外，值得注意的是，各种归纳逻辑编程范式 [3, 26]会使用同样不变的“外延”谓词，这些谓词可视为 PyReason 中的“静态”谓词。

语法:
$I(A, \hat{t})$: $[L, U]$
其中， A 可以是原子（命题情形）或谓词（一阶逻辑）， $\hat{t}$要么是解释 I 有效的时间点 $T = t$ ，要么如果解释是静态的，即在所有时间点上保持不变，则为 $\hat{t} = s$。因此，

$$\hat{t} = \begin{cases} s, & \text{if } I(A, \hat{t}) \text{ is static} \\ t, & t \in T \text{ if } I(A, \hat{t}) \text{ is time-variant} \end{cases} \tag{1}$$

注释 $[L, U] \to [0, 1]$（或者，在命题情形下为 $[L, U] \in [0, 0],\ [1, 1]$）。我们通过为原子及其否定分别设置解释，将文字纳入系统。需要注意的是，除静态原子外，不同时间点上的基原子彼此之间不一定相关。例如，时间 1 上的原子“a”可以标注为 $[0.5, 0.7]$，而时间 2 上则标注为 $[0.1, 0.2]$ 。时间点之间没有单调性要求。

示例 2.3（解释）。继续上一个示例，
关于学生注册的初始事实集：
`I(student(john),0) = [1,1]` *(John is enrolled as a student)*
`I(student(mary),0) = [1,1]` *(Mary is enrolled as a student)*
`I(student(phil),0) = [0,0]` *(Phil is not enrolled as a student)*
静态解释可用于表示始终为真的事实，例如：
I(class(english), s) `= [1,1]`（英语是一门在所有时间点均开设的课程）
使用时间标注来描述随时间的变化：
`I(takes(john,math),1) = [1,1]` *(John takes Math class at time* $t = 1$*)*
`I(takes(john,math),5) = [0,0]` *(But is no longer taking Math at* $t = 5$*)*
如果未在 $t = 0$ 中指定，则所有其他解释都将使用 $[0, 1]$ 进行初始化。

逻辑规则。 规则是使由非静态谓词构成的原子发生变化的关键句法结构。历史上，逻辑规则大多由领域专家编写，直到 Apriori [27] 和 FOIL [28] 等早期工作开始从数据中学习关联规则，随后又出现了因果规则挖掘 [29] 和带注释的概率时序逻辑 [24, 30, 31]等规则挖掘技术。近年来，已有关于可微归纳逻辑编程（∂ILP）的研究——这是一种从示例中学习逻辑规则的归纳规则学习方法 [3, 16, 32]。在下方列表中， $UnaSet$ 和 $BinSet$ 分别是与规则相关的一元谓词和二元谓词的任意集合，而 $pred$ 始终是一个非静态谓词。请注意，假定体部中的原子总数为 n （涵盖所有不同的合取）。符号 $\exists_k$ 表示至少存在 k 个常量，使得后续的逻辑句子得到满足。

1. 用于在单个常量或边内进行推理的基规则：$pred(c)$：$f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \bigwedge_{pred_i \in UnaSet} pred_i(c)$：$x_i pred(c, c')$：$f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \bigwedge_{pred_i \in BinSet} pred_i(c, c')$：$x_i$

2. 用于在单个常量或边内进行推理的全称量化非地面规则：$\forall X$：$pred(X)$：$f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \bigwedge_{pred_i \in UnaSet} pred_i(X)$：$x_i$
$\forall X, X' \; s.t. (X, X') \in E$：$pred(X, X')$：$f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \bigwedge_{pred_q \in BinSet} pred_q(X, X')$：$x_q \wedge \bigwedge_{pred_r \in UnaSet} pred_r(X)$：$x_r \wedge \bigwedge_{pred_s \in UnaSet'} pred_s(X')$：$x_s$

3. 用于跨边进行推理的全称量化非地面规则：$\forall X$：$pred(X)$：$f(x_1, \ldots, x_n) \leftarrow_{\Delta t} \exists_k X'$：$rel(X, X')$：$[1,1] \wedge \bigwedge_{pred_q \in BinSet} pred_q(X, X')$：$x_q \wedge \bigwedge_{pred_r \in UnaSet} pred_r(X)$：$x_r \wedge \bigwedge_{pred_s \in UnaSet'} pred_s(X')$：$x_s$

4. 头部中含有基于规则的量词的非地面规则：$pred(X)$：$[A_s(l_1, l_2, \ldots, l_n) \; A_s(u_1, u_2, \ldots, u_n)] \leftarrow \bigwedge_{X_i \text{ 满足条件 } (X, X_i) \in E} pred'(X, X_i)$：$[l_i, u_i]$ 这里，$A^k_{s,m}(S)$ 可以是定义在集合 S 上的基于规则的量词，使得，$A^k_{s,m}(S)$ 是集合 S 中的最高值。

示例 2.4（逻辑规则）。对于这个延续示例，我们可以基于上述格式构造一些有趣的规则：

1. $promoted(X)$：$[T(l_1, l_2) \; U(u_1, u_2)] \leftarrow_{\Delta t=1} student(X)$：$[l_1, u_1] \wedge gpa(X)$：$[l_2, u_2]$其含义是：“如果 X 是一名边界为 $[l_1, u_1]$ 的学生，并且其平均绩点的边界为 $[l_2, u_2]$，那么在下一个时间步，X 很可能会晋升，其边界由 $[l_1, u_1]$ 和 $[l_2, u_2]$ 的函数给出。”

此处，T 可以是一个 *T*范数。一些常见的 *t*-范数示例包括：*a)* 最小值：$T(a, b) = T_{min}(a, b) = min(a, b)$ *b)* 乘积：$T(a, b) = T_{prod}(a, b) = a \cdot b$ *c)* 卢卡西维茨：$T(a, b) = T_{luk}(a, b) = max(0, a + b - 1)$ *PyReason* 还支持其他常见的逻辑函数，如 $T - conorm$，以及代数函数，如 $max, min, average$等。

2. $\forall X, Y \; expertise(X, Y)$：$[0.6 * L, 1] \leftarrow_{\Delta t=0} grade[X, Y]$：$[L, 1] \wedge student(X)$：$[1, 1] \wedge class(Y)$：$[1, 1]$其含义是：“如果 X 是一名在课程 Y 中取得成绩 $[L, 1]$ 的学生，那么我们可以通过在单个注释 $[L, 1]$上定义注释函数 $[0.6 * L, 1]$，来估算X在学科 Y 方面的专业知识。”

3. $gpa(john)$: $[\frac{x_1+x_2}{2}, 1] \leftarrow_{\Delta t=0} \exists_{i=2} C_i \in \mathcal{C}$: $class(C_i)$: $[1,1] \wedge takes(john, C_i)$: $[1,1] \wedge grade(john, C_i)$: $[x_i, 1]$其含义是："如果 $john$ 修读两门课程并取得相应成绩，那么可以使用给定存在量化基规则头部中的代数函数 avg 来计算其 gpa 。"

4. $friend(S, S')$: $[1,1] \leftarrow_{\Delta t=2} takes(S, C)$: $[1,1] \wedge takes(S', C)$: $[1,1] \wedge class(C)$: $[1,1]$一条带有时间扩展的命题规则，其含义是："如果两名学生 S 和 S' 上同一门课程 C，那么经过两个时间步后，他们会成为朋友。"

5. $\forall S, S', S''$ $friend(S, S'')$: $[1,1] \leftarrow_{\Delta t=1} friend(S, S')$: $[1,1] \wedge friend(S', S'')$: $[1,1]$ 一条与数学中的结合律类似的全称量化非地面规则，其含义是："拥有共同的朋友 S' 会促使两个人 S和 S''成为朋友。"

演绎不动点算子。 演绎过程的核心是不动点算子（记作 Γ），此前已证明，该算子能够在 [6, 7] 中生成逻辑程序（规则和事实）所蕴含的所有原子；这些结果随后又扩展到了时间语义[9, 8]。值得注意的是，这是不动点的精确计算，因此能够提供与逻辑程序相关联的最小模型，从而可以轻松检查任意公式是否被蕴含。此外，结果也完全可解释：对于任何蕴含查询，我们都可以获得导出该结果的一系列推理步骤。这与其他无法解释演绎结果的框架 [18] 有显著区别；不过，关键差异在于，PyReason 中实现的推理框架能够进行精确且高效的多项式时间推理，而其他框架的推理过程则难以处理。

示例 2.5（不动点算子(Γ)）。考虑除了之前指定的初始解释之外，我们还有以下一组初始解释：

I(takes(john,english),1) = I(takes(john,english),2) = [1,1] I(takes(mary,english),2) = I(takes(mary,english),3) = [1,1](John 在 t=1, 2 修读英语，Mary 在 t=2,3 修读英语) I(friend(mary,phil),s) = [1,1](在所考虑的整个时间段内，Mary 和 Phil 都是朋友)

并且我们考虑规则集 $\boldsymbol{R}$ 由上文的规则*4*和*5*构成。我们初始化：∀*S,S' I(friend(S,S'),0) = [0,1] (*所有 $friend$ 关系均初始化为未知*)*，然后更新：*I(friend(mary,phil),s) = [1,1](*来自初始解释*)*

在 *T=0* 时应用 Γ 和 *1*，不会使 $\boldsymbol{I}$ 发生变化，因为没有任何规则被触发。在 *T=2* 时，规则*4*被触发，产生以下基实例化：$friend(john, mary)$: $[1,1] \leftarrow_{\Delta t=2} takes(john, english)$: $[1,1] \wedge takes(mary, english)$:$[1,1] \wedge class(english)$: $[1,1]$

$friend(mary, john)$: $[1,1] \leftarrow_{\Delta t=2} takes(mary, english)$: $[1,1] \wedge takes(john, english)$: $[1,1] \wedge class(english)$: $[1,1]$这会导致 $\boldsymbol{I}$ 在 $T = 4$ 时发生变化，因为 $\Delta t = 2$ 对于上述规则成立，并且该规则在 *T=2* 时被触发。*I(friend(john,mary),4)* `= [1, 1]` *I(friend(mary,john),4)* `= [1, 1]`在 *T=3* 时，由于 $\boldsymbol{I}$ 仍未改变，应用 Γ 不会触发任何规则。在 *T=4* 时，应用 Γ 并结合更新后的解释，会触发已落地规则*5*，如下所示：$friend(john, phil)$: $[1,1] \leftarrow_{\Delta t=1} friend(john, mary)$: $[1,1] \wedge friend(mary, phil)$: $[1,1]$ 结果为：*I(friend(john, phil),5)* `= [1, 1]`

上述内容展示了 *PyReason* 如何通过精确应用不动点算子*(*Γ*)*进行逻辑推理。在此示例中，我们能够追踪解释 *I(friend(john, phil),t)* 如何随时间变化，以及哪些规则导致了这些变化。这表明该过程是完全可解释的，并且可以应用于新兴的神经符号应用。

常量-谓词类型检查约束。 降低复杂度并加快推理过程的关键在于类型检查。我们利用知识图谱中普遍存在的稀疏性，在基元化过程中显著缩小搜索空间。我们注意到，图通常包含不同类型的节点，而谓词通常仅针对特定类型的常量定义。在初始化解释时，类型检查会将这一点纳入考虑，只为彼此兼容的谓词-常量对的子集创建基原子。不过需要注意的是，这是一项可选功能，因为在某些应用中可能无法获得此类信息。

示例 2.6（常量-谓词类型检查）。 在接续的示例中，我们看到谓词 *student*、*gpa* 和 *promoted* 仅限于 *student* 类型的常量。同样，谓词 *class* 和 *difficulty* 仅适用于常量 *english* 和 *math*。类型检查确保我们不会考虑诸如 *student(english)* 或 *class(phil)* 这样的基原子。

二元谓词 *takes(S,C)* 也是如此：第一个变量总是以学生类型的常量为基，第二个变量则以课程类型的常量为基。即使在这个微型示例中，类型检查也将待考察的地面原子数量从 *25* 个减少到仅 *6* 个，降幅达 *76%*。随着图的规模增大、稀疏度提高，这些收益会显著降低复杂度。

检测并解决不一致性。以下情况可能会出现不一致性：

1. 对于某个地面原子，新解释被分配了一个注释 $[L', U']$，该注释不是当前解释 $[L, U]$ 的子集（我们假设 $L \leq U$）。也就是说，如果 $U < L'$ 或 $U' < L$。2. 当一个原子与其否定之间出现不一致性时，例如“a”和“not a”；或者当互补谓词之间出现不一致性时，例如“$bachelor(X)$”和“$married(X)$”，它们无法同时成立。例如，文字 A 的注释为 $[L_1, U_1]$，而文字 B 是文字 A 的否定，其

注释为 $[L_2, U_2]$。不动点算子尝试为文字 A 分配 $[L'_1, U'_1]$，并为文字 B 分配 $[L'_2, U'_2]$。但新的边界不一致，即 $L'_1 > 1 - L'_2$ 或 $U'_1 < 1 - U'_2$。

PyReason 会标记不动点算子执行期间出现的所有此类不一致性，并将其报告出来。此外，由于不动点算子会提供可解释的轨迹，用户可以看到不一致性的确切原因。作为一项实用的附加功能，PyReason 提供了一个选项，可将任何已识别的不一致性对应的注释重置为 $[0, 1]$，并在推理过程剩余阶段将该原子设为静态。这样一来，此类不一致性便不会进一步传播。这些初始功能为更高级的一致性管理技术奠定了坚实基础，例如实现局部一致性或对初始逻辑程序进行迭代松弛。

示例 2.7（检测并解决不一致性。） 假设我们有以下先验知识：

I(takes(phil,数学), 4) `= [1,1]` *I(takes(mary,数学), 4)* `= [1,1]` *I(朋友(phil,mary), 5)* `= [0,0]`

然而，以下逻辑规则的基元化为 $S \leftarrow phil, S' \leftarrow mary, C \leftarrow math$：
$friend(S, S')$: $[1,1] \leftarrow_1 takes(S, C)$: $[1,1] \wedge takes(S', C)$: $[1,1]$ 在 $t = 4$被触发。
由此得出：

I(朋友(phil,mary), 5) `= [1,1]`

但显然，这是一个不一致性，因为 *I(朋友(phil,mary), 5)* 不可能同时既是 $[0, 0]$又是 $[1, 1]$。因此，我们得出结论：这两种解释中至少有一种是错误的。如果无法确定哪一种是正确的，我们可以通过设置以下内容来消解这一逻辑不一致性：

`I(friend(phil,mary), s) = [0,1]` *at* $t = 5$.

3. 实现

我们致力于打造一个现代化的基于 Python 的框架，以支持可扩展且正确的推理。我们支持通过便捷的 *Graphml* 格式输入图数据，这种格式常用于知识图谱架构。我们使用 Networkx Python 库加载图数据并与之交互。目前，我们正在推进对 Neo4j 的直接支持。初始条件和规则以 YAML 格式输入，我们采用节省内存的实现技术，准确捕捉语义结构。我们使用开源 JIT 编译器 Numba，将许多关键操作转换为快速、优化的机器码，同时允许用户通过 Python 以及上述前端进行交互。我们的实现支持 CPU 并行，这一点已通过在多 CPU 机器上运行的实验得到验证。

我们的软件将解释存储在嵌套字典中。为提高计算效率并方便使用，我们的软件允许指定一系列时间点 $T = t_1, t_2, ...$，而不是单个时间点 t，并且解释 I 在这些时间点范围内保持有效。为减少内存需求，任意时刻只存储一组解释（当前解释）。不过，可以使用规则轨迹获取过去的解释；规则轨迹会保留每个解释的变更历史，以及导致每次变化的相应已基元化逻辑规则

表 1 Honda 网络：大流行病造成一国行业中断后，如何可能波及全球

公司		时间 t= 全球各地受影响的公司							受影响公司的比例		
基于	数量	0	1	2	3	4	···	38	初始	最终	**变化**
USA	1599	1599	1965	2057	2203	2313	···	**3336**	14.68	30.75	**16.07**
台湾	603	603	644	**647**	647	647	···	647	5.54	5.94	**0.40**
澳大利亚	128	128	**131**	131	131	131	···	131	1.18	1.21	**0.03**

，以及导致每次变化的相应已基元化逻辑规则。规则轨迹让我们的软件完全可解释，因为每次推理都可以追溯到引发该推理的一连串规则。

MANCaLog [9] 展示了在规范模型和非规范模型中使用不动点算子。通过在每个时间步重新计算解释，我们不仅大幅减少了内存需求，还同时支持规范和非规范情形。得益于这一设计，计算时间的增幅极小。

此外，我们在 [33] 方面取得了重大进展：支持静态谓词，并内置了非图推理和类型检查功能，详情见第2节。

我们的实现可在线获取，具体信息见第1 节；详细的伪代码见补充材料。

4. 实验

4.1. 本田买方-供应商数据集

我们在本田买方-供应商网络 [21]上开展实验。该数据集（网络）包含 10,893 家公司（节点），以及它们之间的 47,247 条买方-供应商关系（边）。

我们设计了一个用例，假设某个国家的所有公司的运营都受到干扰，并观察这可能对世界各地的公司造成的影响。我们认为，这与 COVID-19 疫情期间全球面临的供应链问题类似。在测试中，我们使用以下逻辑规则；在实际应用中，该规则可以通过学习获得，也可以由专家提供。

$disrupted\,(Buyer)\colon\ [1,1] \leftarrow_{\Delta t=1} \forall_k supplies\,(Sup_k,\ Buyer)\colon\ [1,1],\ \exists_{k/2} disrupted\,(Sup_k)\colon\ [1,1]$该规则指出，如果某家公司至少 50% 的供应商在上一个时间步完全中断，那么该公司在特定时间步就会受到干扰。我们针对三个不同的国家（美国、台湾和澳大利亚）开展了这项实验，这些国家在数据集中所占的公司比例各不相同。我们不预先设定推理步骤的数量，而是让扩散过程持续运行，直至收敛（以粗体标示）。结果见表 1。

为了测试我们的方法是否具备可扩展性，我们使用两条推理规则：如果某家公司的一家或多家供应商在前一个时间步完全中断，或者至少 50% 的供应商的产能中断程度达到至少 50%，则该公司在特定时间步会受到影响。我们针对不同的图规模和不同的时间步数量开展了这项实验，以

表2我们的框架的可扩展性

节点（N）	边（E）	属性总数	密度	时间步	运行时间（秒）	内存（MB）
1000	410	5012	4.10×10^{-4}	2	0.36	4.9
				5	0.42	1.8
				15	0.34	0.1
2000	1640	13269	4.10×10^{-4}	2	0.43	1.2
				5	0.55	2.1
				15	0.81	8.2
5000	10244	57852	4.10×10^{-4}	2	1.54	17.2
				5	1.84	16.0
				15	3.38	54.6
10000	41034	197752	4.10×10^{-4}	2	4.83	80.3
				5	6.29	60.3
				15	12.34	210.8

展示我们软件的扩展能力，见表2。

结果表明，在较大范围内，运行时间和内存占用几乎保持不变，随后随着网络规模的增加呈次线性增长。

4.2. Pokec 社交媒体数据集

Pokec 是一个颇受欢迎的斯洛伐克社交网络，该数据集 [22] 包含 160 万人（节点）的个人信息，例如性别、年龄和宠物（属性），以及他们之间的 3060 万条连接（边）。

我们借鉴广告行业的做法来设计用例。我们将拥有宠物的人口中的一小部分视为一家宠物食品公司的客户。该公司必须利用 Pokec 数据，在人口中识别相关目标对象。一个切实可行的策略可以用以下两条逻辑规则来表示：

1. $\forall X, Y relevance(X) : [0.6, 1] \leftarrow_{\Delta t=1} relevance(Y) : [1, 1] \wedge friend(X, Y) : [1, 1]$相关目标对象或现有客户的朋友（始终是相关对象）至少有 60% 的相关性。 2. $\forall X, Y relevance(X) : [1, 1] \leftarrow_{\Delta t=1} relevance(Y) : [1, 1] \wedge friend(X, Y) : [1, 1] \wedge hasPet(X, P) : [1, 1] \wedge hasPet(Y, P) : [1, 1]$ 如果某人拥有与相关目标对象相同种类的宠物（狗、猫等），那么该相关目标对象的朋友就完全相关。

扩散过程在经过 8 个时间步后收敛，耗时 42 分钟，使用了 58.36 GB 内存——这进一步展现了我们框架的可扩展性。结果如表3所示。

推理过程完全可解释，用户可以使用 规则轨迹（PyReason 的可选输出），来确定导致每种解释发生变化的逻辑规则。上一个实验中的规则轨迹示例如表 4所示。

所有实验均在一个配备 96 个虚拟 CPU（48 个核心）和 384GB 内存的 AWS EC2 容器上进行。

表 3 Pokec 社交媒体：品牌如何利用消费者数据识别潜在客户

			广告目标受众	
人口规模	现有客户	时间步	完全相关	部分相关
1,632,803	2,308	0	2,308	0
		1	2,596	39,836
		2	2,657	47,405
		3	2,679	49,174
		4	2,690	50,046
		5	2,692	50,412
		6	2,693	50,455
		7, 8, ⋯	2,693	50,608

表 4

标签相关性的单个节点规则轨迹。应用上方的规则 1 导致了从 $[0, 1]$ 到 $[0.6, 1]$的首次变化，随后由于规则 2 的触发，更新为 $[1, 1]$。此外，还提供了用于规则子句实例化的节点和边ID列表。

t	Old 界限	New 界限	Rule 已触发	子句-1	子句-2	子句-3	子句-4
1	[0.0,1.0]	[0.6,1.0]	规则 1_	[‘354455’]	[(‘354365’, ‘354455’)]		
2	[0.6,1.0]	[1.0,1.0]	规则 2_	[‘354455’, ‘718503’]	[(‘354365’, ‘354455’), (‘354365’, ‘718503’)]	[(‘718503’, ‘猫’)]	[(‘354365’, ‘猫’)]

5. 相关工作

在第 1 节中，我们讨论了 PyReason 如何在 Prolog [19], Epilog [20]和 Datalog 等早期现代逻辑程序设计语言的基础上，通过支持注释进行扩展 [34]。近期的神经符号框架在学习或修改逻辑程序、使其与历史数据保持一致并提高对噪声的鲁棒性方面展现出巨大潜力。许多此类框架都依赖底层的可微模糊一阶逻辑。例如，逻辑张量网络 [1] 使用模糊算子的可微版本来组合地面原子和非基原子命题，而逻辑神经网络 [2]则将实数区间与原子命题关联起来，并使用特殊的参数化算子。与此同时，诸如可微归纳逻辑编程 [3] 之类的归纳方法会从数据中学习模糊逻辑程序（使用乘积 t-范数），其方式基于模板规则结构，并支持递归和多步推理。在 [17], 逻辑神经网络中，学习到的规则可被精确地解释。这里同样使用梯度下降来训练网络参数。在过去两年中，神经符号领域出现了两种广受欢迎的范式。逻辑张量网络（LTN） [1] 通过模糊实值逻辑扩展神经架构。逻辑神经网络（LNN） [2] 提供了一种带有参数化算子的神经符号框架，并支持逻辑中的开放世界推理。如前所述，两者都可以视为带注释逻辑的子集。因此，PyReason 可用于对这两个框架中的逻辑进行推理，

除了提供基于图的推理和时序推理等关键能力外，这些能力目前在这些框架所采用的逻辑中尚不存在。

在各种神经符号框架 [35, 2, 1], 的前向传播中，以及在后续任务（例如蕴含、溯因推理、规划等）中，都需要演绎过程。PyReason 旨在提供这一精确能力。已有研究表明，广义注释程序 [6] 能够捕获各种实值逻辑、时序逻辑和模糊逻辑，因为它将逻辑原子与格结构中的元素相关联，而不是与标量值相关联。因此，它能够捕获上述所有逻辑，同时由于格的单调性而保留多项式时间演绎能力。格结构的使用使我们能够将逻辑构造与区间相关联，从而实现开放世界推理。在我们最近的工作中 [7], ，我们对 [6] 进行了扩展，使注释能够采用下格结构。这使该框架能够捕获 LNN [2] 和 MANCALog [9] 等用于基于图的推理的范式。然而，该工作仅证明了下格情况下与 [7]定理相对应的定理，并未提供实现或实验结果。

通过支持广义注释逻辑及其各种扩展，PyReason 能够实现独立于学习过程的系统设计。因此，一旦神经符号学习过程基于数据创建或修改了逻辑程序，PyReason 就能高效地回答演绎查询（包括蕴含和一致性查询），并支持更复杂的推理，例如溯因推理或规划。

如今，知识图谱对于表示用于推理和分析的数据至关重要。近期有关知识图谱创建的研究 [36, 37] 提出了自动将概念模型转换为 GraphML 格式知识图谱的方法，应用于企业架构及广泛的其他领域。PyReason 支持 GraphML 格式，可作为分析通过这些平台之一获取的知识图谱的有效工具。

6. 结论与未来工作

本文介绍了 PyReason：一种可解释的推理软件，支持带注释、开放世界、实值、基于图的时态逻辑。我们的现代化实现扩展了成熟的广义注释逻辑框架，以支持在大型知识图谱和扩散模型上进行可扩展且高效的推理。目前，我们正在开展多项扩展工作，包括添加更多用于规范检查的时序逻辑算子、通过归纳从数据中学习规则，以及利用推理过程在非静态图中创建新知识（例如，添加节点和边）。我们还将探索如何将 PyReason 与 LTN [1], 和 LNN [2]结合使用。为支持此类框架，我们将致力于增加符号落地功能 [38], ，利用 LTN 等框架训练过程的结果。最后，我们还计划扩展 PyReason，使其能够充当基于强化学习的智能体的模拟器。

致谢

作者获得富尔顿工程学院的内部资助，本研究的部分工作获得美国陆军小企业技术转移计划的支持

美国陆军研究办公室或合同编号W911NF-22-P-0066项下。

参考文献

[1] S. Badreddine、A. d’Avila Garcez、L. Serafini、M. Spranger，《逻辑张量网络》，《人工智能》303（2022年）103649。[2] R. Riegel、A. Gray、F. Luus、N. Khan、N. Makondo、I. Y. Akhalwaya、H. Qian、R. Fagin、F. Barahona、U. Sharma、S. Ikbal、H. Karanam、S. Neelam、A. Likhyani、S. Srivastava，《逻辑神经网络》，2020年。[3] R. Evans、E. Grefenstette，《从噪声数据中学习解释性规则》，《人工智能研究期刊》61（2018年）1–64。[4] M. Ma、J. Gao、L. Feng、J. Stankovic，STLnet：由信号时序逻辑约束的多变量循环神经网络，载于：神经信息处理系统进展，第33卷，Curran Associates, Inc.，2020年，第14604–14614页。[5] P. Sen、B. W. Carvalho、I. Abdelaziz、P. Kapanipathi、S. Roukos、A. Gray，《用于结合嵌入表示和规则进行知识库补全的逻辑神经网络》，载于：自然语言处理实证方法会议，2022年，第3863–3875页。[6] M. Kifer、V. Subrahmanian，《广义标注逻辑编程的理论及其应用》，《逻辑程序设计期刊》12（1992年）335–367。[7] P. Shakarian、G. I. Simari，《广义注释逻辑的扩展及一种等价神经架构》，载于：2022 TransAI，2022年，第63–70页。[8] P. Shakarian、G. I. Simari、D. Callahan，《复杂网络推理：一种逻辑程序设计方法》，《逻辑程序设计理论与实践》13（2013年）。[9] P. Shakarian、G. I. Simari、R. Schroeder，MANCALog：一种用于多属性网络级联的逻辑，载于：自主智能体与多智能体系统国际会议，AAMAS，2013年，第1175–1176页。[10] U. Höhle，《模糊拓扑的概率一致化》，《模糊集与系统》（1978年）。[11] C. Alsina、E. Trillas、L. Valverde，《关于模糊集理论的若干逻辑联结词》，《数学分析与应用期刊》93（1983年）15–26。[12] P. Vojtáš，《模糊逻辑编程》，《模糊集与系统》124（2001年）361–370。[13] M. H. Van Emden、R. A. Kowalski，《作为程序设计语言的谓词逻辑语义》，《ACM期刊》23（1976年）733–742。[14] M. Fitting，《逻辑程序设计中的双格》，纽约市立大学莱曼学院数学与计算机科学系，1990年。[15] P. Shakarian、M. Broecheler、V. Subrahmanian、C. Molinaro，《使用广义注释程序解决社交网络扩散优化问题》，《ACM计算逻辑汇刊》14（2013年）。[16] H. Shindo、M. Nishino、A. Yamamoto，《面向结构化样例的可微归纳逻辑编程》，载于：AAAI人工智能会议，2021年，第5034–5041页。[17] P. Sen、B. W. S. R. d. Carvalho、R. Riegel、A. Gray，《利用逻辑神经网络进行神经符号归纳逻辑程序设计》，AAAI人工智能会议（2022年）。[18] P. Hohenecker、T. Lukasiewicz，《使用深度神经网络进行本体推理》，载于：《人工智能研究期刊》，第68卷，2020年，第503–540页。

[19] A. Colmerauer, 《Prolog与无限树》, 《逻辑程序设计》16（1982）2。[20] L. K. Schubert、C. H. Hwang, 《情景逻辑遇见小红帽：一种全面、自然的语言理解表示》, 《自然语言处理与知识表示：知识的语言与语言的知识》（2000）。[21] T. Yan、T. Y. Choi、Y. Kim、Y. Yang, 《枢纽供应商理论：从网络视角看关键供应商》, 《供应链管理杂志》51（2015）52–66。[22] L. Takac、M. Zabovsky, 《公共社交网络中的数据分析》, 载于：国际科学会议暨国际研讨会“当代创新趋势”, 2012。[23] A. Dekhtyar、M. I. Dekhtyar、V. S. Subrahmanian, 《时序概率逻辑程序》, 载于：国际逻辑程序设计会议，1999，第109–123页。[24] S. Khuller、M. V. Martinez、D. S. Nau、A. Sliva、G. I. Simari、V. S. Subrahmanian, 《计算行动概率逻辑程序的最可能世界：对1030,000个世界的可扩展估计》, 《数学年刊与人工智能》51（2007）295–331。[25] P. Shakarian、A. Parker、G. Simari、V. V. S. Subrahmanian, 《带注释的概率时序逻辑》, 《ACM计算逻辑汇刊》12（2011）。[26] M. V. França、G. Zaverucha、A. S. d’Avila Garcez, 《使用底子句命题化与人工神经网络进行快速关系学习》, 《机器学习》94（2014）。[27] R. Agrawal、H. Mannila、R. Srikant、H. Toivonen、A. I. Verkamo等, 《快速发现关联规则》, 《知识发现与数据挖掘进展》12（1996）307–328。[28] J. R. Quinlan, 《从关系中学习逻辑定义》, 《机器学习》5（1990）。[29] A. Stanton、A. Thart、A. Jain、P. Vyas、A. Chatterjee、P. Shakarian, 《因果关系挖掘：关于伊斯兰国的数据驱动研究》, 载于：ACM SIGKDD知识发现与数据挖掘国际会议，2015，第2137–2146页。[30] P. Shakarian、A. Parker、G. Simari、V. V. Subrahmanian, 《带注释的概率时序逻辑》, 《ACM计算逻辑汇刊》（TOCL）12（2011）1–44。[31] P. Shakarian、G. I. Simari、V. Subrahmanian, 《带注释的概率时序逻辑：近似不动点实现》, 《ACM计算逻辑汇刊》（TOCL）13（2012）1–33。[32] P. Sen、B. W. de Carvalho、R. Riegel、A. Gray, 《结合逻辑神经网络的神经符号归纳逻辑程序设计》, 载于：AAAI人工智能会议，2022年。[33] J. N. Paredes、G. I. Simari、M. V. Martinez、M. A. Falappa, 《通过知识与数据混合驱动系统检测社交平台中的恶意行为》, 《未来一代计算机系统》125（2021）232–246。[34] S. Abiteboul、R. Hull、V. Vianu, 《数据库基础》, 第8卷，Addison-Wesley，Reading，1995。[35] Z. Yang、A. Ishay、J. Lee, 《Neurasp：将神经网络融入答案集程序设计》, 载于：国际人工智能联合会议，IJCAI, 2020。[36] M. Smajevic、D. Bork, 《从概念模型到知识图谱：通用模型转换平台》, 载于：2021年ACM/IEEE模型驱动工程语言与系统国际会议伴会（MODELS-C）, IEEE，2021，第610–614页。[37] P.-L. Glaser、S. J. Ali、E. Sallinger、D. Bork, 《基于模型构建企业架构知识图谱》, 载于：《企业设计、运营与计算：第26届国际会议》, Springer，2022年，第57–73页。[38] S. Harnad, 《符号接地问题》, 《物理学D：非线性现象》42（1990）。

[39] J. W. Lloyd, 《逻辑程序设计基础》, Springer-Verlag New York, Inc., 1987年。

A. 形式语法与语义。

现在，我们重新阐述广义注释逻辑程序（以下简称“GAP”）的定义（见 [6] ），以及我们在软件中纳入的扩展。

在 [6], 中，作者假设存在一个半格 $\mathcal{T}$（不一定是完备的），其序关系为 $\sqsubseteq$。为了支持当代神经符号推理 [2, 3, 7, 16, 17]以及社交网络分析 [9, 8] 的应用，我们将其实现为下半格结构。因此，我们有一个单一元素 $\bot$ 和多个顶元素 $\top_0$、 $\ldots \top_i \ldots \top_{max}$。记号 $height$（$\mathcal{T}$）表示在格中从 $\bot$到某个顶元素的路径上元素的最大数量（包括 $\bot$ 和顶元素）[2]。采用下半格结构可以实现两种理想特性。首先，我们希望像先前工作 [2, 9, 25]中那样，在 $[0, 1]$ 中用实数区间标注原子。其次，它支持对这类区间进行推理；随着算子沿格结构向上推进，不确定性（即对于区间 $[l, u]$ ，其量$u - \ell$）会单调递减。因此，我们定义底元素 $\bot = [0, 1]$ 和一组顶元素 $\{[x, x] \mid [x, x] \subseteq [0, 1]\}$ （见注释[3]）。具体而言，我们设定 $\top_0 = [0, 0]$ 和 $\top_{max} = [1, 1]$。图2展示了这种半格结构的一个示例。

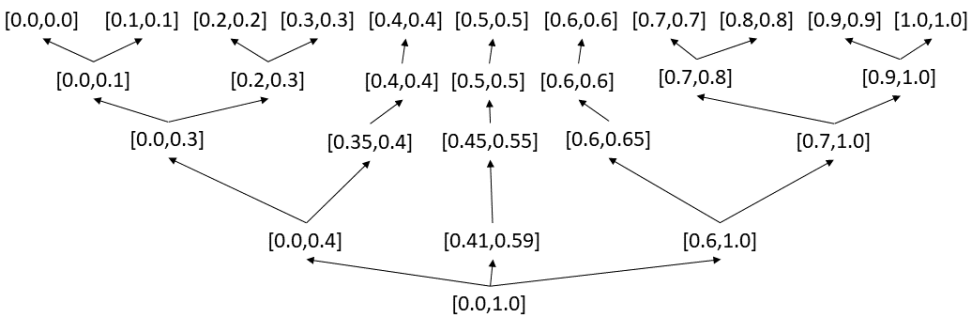

图2： $[0, 1]$中的元素为区间的下半格结构示例。

与 [6], 一样，我们假设存在一个变量符号集合 AVar，其取值范围为 $\mathcal{T}$，以及一个 $\mathcal{F}$ 函数符号集合，其中每个函数符号都具有相应的元数。我们首先定义注释。

定义 A.1（注释）。*(i)* $\mathcal{T} \cup AVar$ 中的任何成员都是注释。*(ii)* 如果 f 是定义在 $\mathcal{T}$上的 n 元函数符号，且 t_1、 ..... 、 t_n 都是注释，那么 $f(t_1, ..., t_n)$ 就是一个注释。

[2]一般而言，我们假设该格由有限离散元素组成。[3]请注意，在使用界半格时，符号“$\sqsubseteq$”会失去其“子集直觉”，因为 $[0, 1] \sqsubseteq [1, 1]$ 在此例中就是如此。

我们定义的一个特定函数是“¬”，它用于 [6] 的语义，以及 [2]中使用的较新基于区间的框架。对于给定的 $[l, u]$，$\neg([l, u]) = [1 - u, 1 - l]$。请注意，我们也在一阶语言中使用符号 $\neg$（遵循 [6]的形式体系）。我们定义一种独立的逻辑语言，其常量属于集合 $\mathcal{C}$，谓词符号由集合 $\mathcal{P}$指定。我们还假设存在一个变量符号集合 $\mathcal{V}$，其取值范围为这些常量；不存在函数符号；并且项和原子按通常方式定义（参见 [39]）。我们假设 $\mathcal{C}$, $\mathcal{P}$, $\mathcal{V}$ 都是离散且有限的。一般而言，我们用大写字母表示变量符号，用小写字母表示常量。与先前的工作类似，[3, 18] 假设 $\mathcal{P}$ 中的所有元素的元数均为 1 或 2，因此分别将它们记作一元谓词 $\mathcal{P}_{una}$ 和二元谓词 $\mathcal{P}_{rel}$。我们还将 $\mathcal{P}$ 的一个子集记作“目标谓词” $\mathcal{P}_{tgt}$，其中可包含二元谓词或一元谓词（$\mathcal{P}_{tgt_rel}$, $\mathcal{P}_{tgt_una}$），前提是它们不是保留字。我们用符号 $\mathcal{L}$ 表示所有基元文字的集合，用 $\mathcal{A}$ 表示所有基原子的集合。现在，我们定义本工作中将使用的 GAP 的句法结构。

定义 A.2（带注释的原子式、否定、文字）。核心语法结构定义如下：

- 带注释的原子式。 **</style>**若 是一个原子且 是一个注释，则 **：** μ 是一个带注释的原子式。
- 带注释的否定。 若 a 是一个原子且 μ 是一个注释，则 $\neg a : \mu$ 是一个带注释的否定。

- 带注释的文字。 统称原子和否定为带注释的文字。

定义 A.3（GAP 规则）。 若 $\ell_0 : \mu_0$、$\ell_1 : \mu_1$、**.....**、$\ell_m : \mu_m$ 均为带注释的文字（且对于所有 i、$j \in 1$、m、$\ell_i \neq \ell_j$），则

$$r \equiv \ell_0 : \mu_0 \quad \leftarrow \quad \ell_1 : \mu_1 \wedge \ldots \wedge \ell_m : \mu_m$$

称为GAP 规则。我们使用记号 $head(r)$ 和 $body(r)$ 分别表示 ℓ_0 和 $\{\ell_1, \ldots, \ell_m\}$。当 $m = 0$ **(**$body(r) = \emptyset$**)** 时，上述 ***GAP*** 规则称为事实。当且仅当规则中没有来自 ***AVar*** 或 $\mathcal{V}$ 的变量时，该 ***GAP*** 规则为基规则。对于基规则 r 和基文字 ℓ，$bodyAnno(\ell, r) = \mu$，使得 $\ell : \mu$ 出现在 r 的体部中。广义带注释程序 Π 是有限个 ***GAP*** 规则组成的集合。

GAP 的形式语义定义如下。请注意，我们将解释的概念扩展为允许将文字映射到注释（而非原子）。不过，我们在每个原子与其否定之间的注释上增加了一项要求，以确保其等价于 [6] 的语义结构。

定义 A.4（解释）。解释 I 是从所有基文字的集合到 $\mathcal{T}$ 的任意映射，满足对于文字 a、$\neg a$，有 $I(a) = \neg(I(\neg a))$。所有解释构成的集合 $\mathcal{I}$ 可通过以下序关系定义偏序：$I_1 \preceq I_2$ 当且仅当对于所有基文字 a，$I_1(\ell) \sqsubseteq I_2(\ell)$。$\mathcal{I}$ 在 $\preceq$ 序关系下构成完备格。

现在我们来介绍满足关系：

定义 A.5（满足）。一个解释 I 满足一个基文字 $\ell:\ \mu$，记作$I \models \ell:\ \mu$，当且仅当 $\mu \sqsubseteq I(\ell)$。I 满足基 *GAP* 规则

$$\ell_0 : \mu_0 \quad \leftarrow \quad \ell_1 : \mu_1 \wedge \ldots \wedge \ell_m : \mu_m$$

（记作 $I \models \ell_0:\ \mu_0 \leftarrow \ell_1:\ \mu_1 \wedge \ldots \wedge \ell_m:\ \mu_m$）当且仅当以下任一条件成立

1. I 满足 $\ell_0:\ \mu_0$，或2. 存在一个 $1 \leq i \leq m$，使得 I 不满足 $\ell_i:\ \mu_i$。

I 满足一个非基文字或规则，当且仅当 I 满足它的所有基实例。

我们称解释 I 是程序 Π 的模型，如果它满足 Π 中的所有规则。同样，如果存在某个 I 是 Π 的模型，我们就称程序 Π 是一致 的。我们称 Π 蕴含 $\ell:\ \mu$，记作 $\Pi \models \ell:\ \mu$，当且仅当对于每个满足 $I \models \Pi$ 的解释 I，都有 $I \models \ell:\ \mu$。如 [6], 所示，我们可以为任意 GAP Π 关联一个不动点算子，它将解释映射到解释。

定义 A.6. 设 Π 是任意 *GAP*，I 是一个解释。将解释映射到解释的映射$\mathbf{T}_\Pi$ 定义如下：

$$\mathbf{T}_\Pi(I)(\ell_0) = \mathbf{sup}(annoSet_{\Pi,I}(\ell_0)),$$

其中 $annoSet_{\Pi,I}(\ell_0) = \{I(\ell_0)\} \cup \{\mu_0 \mid \ell_0:\ \mu_0 \leftarrow \ell_1:\ \mu_1 \wedge \ldots \wedge \ell_m:\ \mu_m$ 是 Π 中某条规则的一个基实例，并且对于所有 $1 \leq i \leq m$，我们都有 $I \models \ell_i:\ \mu_i\}$

[6] 的关键结果告诉我们，最小不动点($\mathbf{T}_\Pi$) 精确地刻画了 Π 的基原子逻辑后果。我们证明，即使注释基于较低格，只要满足 Π 一致这一条件，这一结论仍然成立。在 [6], 中，作者还将 $\mathbf{T}_\Pi$ 的迭代 定义如下：

- $\mathbf{T}_\Pi \uparrow 0$ 是将 $\bot$赋给所有基文字的解释。
- $\mathbf{T}_\Pi \uparrow (i+1) = \mathbf{T}_\Pi(\mathbf{T}_\Pi \uparrow i)$.

对于每个地面 $\ell \in \mathcal{L}$，集合 $\Pi(\ell)$ 是 Π 中包含事实在内且头部含有 ℓ 的基规则子集。我们用符号 m_ℓ 表示 $\Pi(\ell)$ 中规则的数量。对于给定的基规则，我们用符号 $r_{\ell,\ i}$ 表示它是头部含有原子 ℓ 的第 i 条规则。

B. 作出较低格假设时结果的形式证明。

请参阅 [7]。

表 5本田买方-供应商网络概览

公司（节点）	10,893
买方-供应商关系（边）	47,247
全球行业分类标准（GICS）类别	67
边关系类型	4

C. 供应链实验的更多详情

表格 5 总结了该数据集的内容。7,396 家公司列有唯一的 GICS，而 3,497 家公司未列出 GICS。

为了更深入地了解买方-供应商网络的结构，我们映射了一个行业及其第一层和第二层供应商，如图所示3。图中显示的每个节点都包含一个属性 名称，用于描述行业的名称；连接这些行业的每条边都包含一个属性成本，用于描述由一条边连接的两个特定行业之间的利润关系。

图 4 展示了部分数据，呈现供应网络的复杂性。该图有助于识别网络中连接大块区域与其他主要区域的若干关键节点——这些节点对运营至关重要。

图 5 展示了各种行业类型的分布。图 6展示了数据集中存在的不同边关系。

D. 社会网络实验的更多详情

表 6 概述了该数据集，模式如图 7所示。图中列出了边关系类型。

图密度表示图中现有边数与图所能包含的最大边数之比。实际上，图通常是稀疏的（而非稠密的）。为了



图3： 展示网络链接连接及属性标签的快照

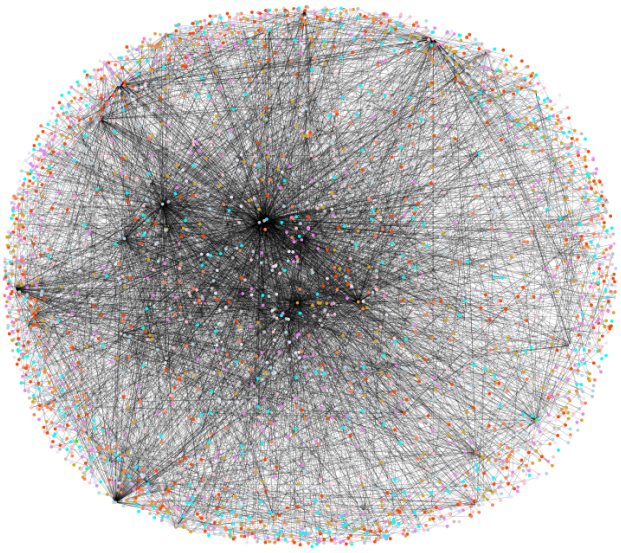

Figure 4: 本田供应链网络。各节点和边根据类型以不同颜色表示 .

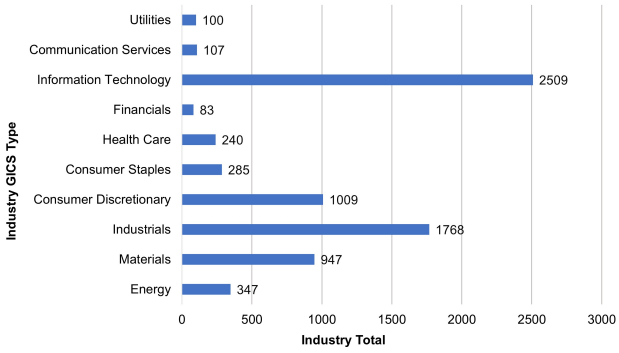

图5: 网络中节点按 GICS 类型的分布。

测试我们的方法在不同图密度下的扩展能力，我们在修改数据集密度的同时重新运行表2中的实验。表7中的结果显示，Honda 数据的密度约为 Pokec 网络的 40 倍，但相关实验仅需约 3 倍内存，运行时间也仅为其 6 倍。进一步展现了我们框架的扩展能力。

最后，为了测试我们的方法在图中属性数量（因此也包括地面原子数量）方面的可扩展性，我们针对一个简单用例运行了一项实验：先使用原始数据集运行一次，然后再加入属性运行一次。我们基于社交媒体数据将用例定义如下：

$\forall X,\ Y\ \ infected(X){:}\ \ [1,1] \leftarrow_{\Delta t=1} infected(Y){:}\ \ [1,1] \wedge friend(X,\ \ Y){:}\ \ [1,1]$

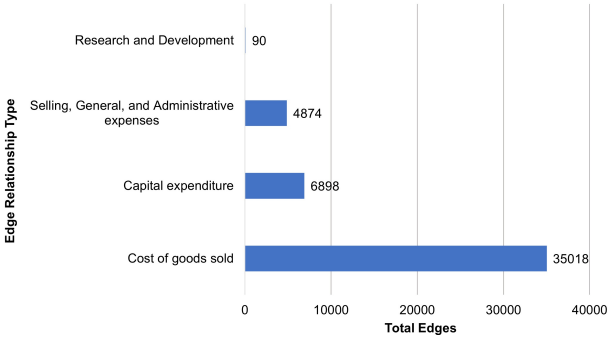

图 6： 数据集中的边关系。

表 6 Pokec 社交媒体数据集概览

节点	1,632,820
边	31,633,113
节点属性类型	5
边关系类型	4

其内容是：“如果某人的任何一位朋友感染了病毒，则该人会感染病毒”。结果如表8所示。

E. 扩展规则轨迹

表4中给出了包含 10 个原子的规则轨迹的较长版本，表9中也有展示。

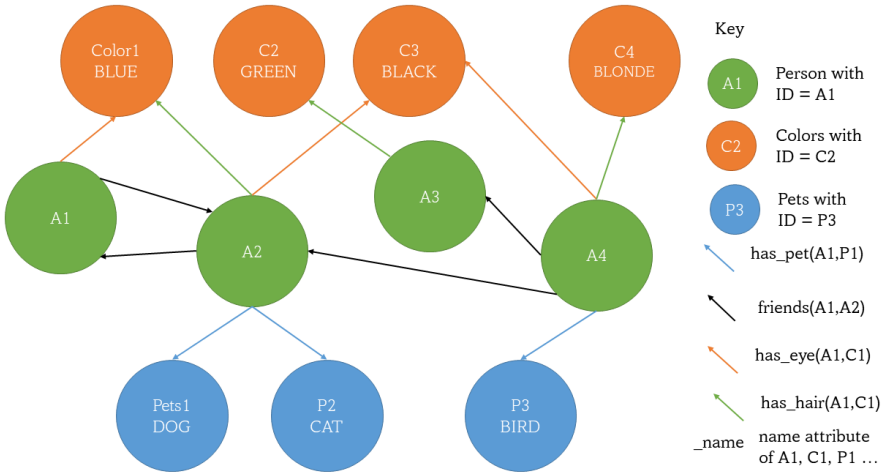

图 7： Pokec 网络的模式

表7图密度对内存和运行时间的影响

节点（N）	密度→ 时间步	Honda 4.10 x 10^{-4} 运行时间（单位：秒）	Honda 4.10 x 10^{-4} 内存（单位：MB）	Pokec 0.11 x 10^{-4} 运行时间（单位：秒）	Pokec 0.11 x 10^{-4} 内存（单位：MB）
10000	2	4.83	80.3	0.70	27.4
	5	6.29	60.3	0.93	19.9
	15	12.34	210.8	1.73	21.2

表 8添加 7 个属性对 5 个时间步内存和运行时间的影响

属性	节点 (N)	边 (E)	基元原子	运行时间（分钟）	内存（GB）
未添加	1,632,803	30,622,563	3,265,606	19.95	42.44
已添加	1,632,820	31,633,113	36,531,539	28.26	59.39
变化	(+17)	(+1,010,550)	(+33,265,933)	(+8.31)	(+16.95)

F. 实现伪代码

算法1 枚举了正在使用的数据结构。算法2 展示了初始状态，而算法3 详细说明了推理过程。在推理期间，解释会按算法4所示进行更新。算法5 和6用于保持逻辑一致性。

算法1所使用的数据结构 1：嵌套字典

$\boldsymbol{I} = [Node/Edge, [Predicate, [Lower, Upper, Static]]]$，用于仅存储当前解释。如果将 $Static$ 设置为 1，则边界：$Lower$，$Upper$ 在程序的其余执行过程中将不再变化。2：列表 $\boldsymbol{L} = [$（$Node/Edge$，$Predicate$，$Lower$，$Upper$，$Static$，at_t）$]$，用于存储事实和推理，然后再用于更新字典。3：列表 **IPL** $= [$（$Predicate_1$，$Predicate_2$）$]$，包含不能同时成立的谓词对，即边界必须成对互补。在命题情形下，如果其中一个谓词为 $true$，另一个必须为 $false$。我们称其为“不一致谓词列表（IPL）”。4：列表 $\boldsymbol{E} = [$（$Node/Edge$，$Predicate$）$]$，包含在程序执行过程中变得不一致的谓词列表。

算法 2程序初始化

1: $\boldsymbol{I}$ 如下：
∀节点/边，仅使用 $type_checking$ 初始化有效谓词。
所有边界均初始化为 [0,1]。 $Static$设为 0。
2: $\boldsymbol{L} \leftarrow [\,]$ ▷ 空列表
事实（包括初始解释）随后被复制到 $\boldsymbol{L}$
3: $t \leftarrow 0$
4: $\boldsymbol{E} \leftarrow [\,]$
5: 输入：扩散时间步数量 T，规则集 $\boldsymbol{R}$

算法 3程序流程

1: **当** $t \leq T$ **时**
2: **对于** i 在 I中，其中（$Static$ 为 $false$）**执行**
3: 将边界重置为 [0,1]
4: **结束循环**
5: $update_req \leftarrow 0$
6: **对于** l 在 $\boldsymbol{L}$中，其中（l（at_t） $== t$）**执行**
7: **如果** 检查_一致性($l \in \boldsymbol{L}$,$l \in \boldsymbol{I}$)**那么**
8: 更新_请求 +=更新_解释($l \in \boldsymbol{L}$,$l \in \boldsymbol{I}$)
9: **else**
10: 解决_不一致性($l \in \boldsymbol{I}$)
11: **如果** $(l, l') \in$ **IPL**, $\forall l'$ **则**
12: 解决_不一致性($l' \in \boldsymbol{I}$)
13: **结束如果**
14: **结束如果**
15: **结束循环**
16: **如果** $update_req$，**则**
17: 应用一次不动点算子($gamma$)。
18: **对** 每个得到的解释**执行**
19: **如果** $Static$ 属于 $false$ 中的 I，**则**
20: 添加到 $\boldsymbol{L}$
21: **结束条件判断**
22: **结束循环**
23: 转到第 5 行。
24: **else**
25: $t \leftarrow t + 1$。
26: **结束条件判断**
27: **结束 while 循环**

表 9表中规则轨迹的更长版本4 的 标签：相关性

t	Γ	Node	界限		Rule 已触发	子句-1	子句-2	子句-3	子句-4
			Old	New					
1	1	1273439	[0.0,1.0]	[1.0,1.0]	规则 2_	[‘835886’]	[(‘1273439’, ‘835886’)]	[(‘835886’, ‘猫’)]	[(‘1273439’, ‘猫’)]
1	1	103308	[0.0,1.0]	[0.6,1.0]	规则 1_	[‘659792’, ‘404372’]	[(‘103308’, ‘659792’), (‘103308’, ‘404372’)]		
2	2	277684	[0.0,1.0]	[1.0,1.0]	规则_2	[‘305645’]	[(‘277684’, ‘305645’)]	[(‘305645’, ‘鱼’)]	[(‘277684’, ‘鱼’)]
2	2	551249	[0.0,1.0]	[0.6,1.0]	规则 1_	[‘377195’]	[(‘551249’, ‘377195’)]		
3	3	861455	[0.0,1.0]	[1.0,1.0]	规则 2_	[‘1450147’]	[(‘861455’, ‘1450147’)]	[(‘1450147’, ‘蜘蛛’)]	[(‘861455’, ‘蜘蛛’)]
3	3	23197	[0.0,1.0]	[0.6,1.0]	规则 1_	[‘25795’]	[(‘23197’, ‘25795’)]		
3	3	757646	[0.0,1.0]	[0.6,1.0]	规则_1	[‘423053’]	[(‘757646’, ‘423053’)]		
4	4	86436	[0.0,1.0]	[1.0,1.0]	规则 2_	[‘743812’]	[(‘86436’, ‘743812’)]	[(‘743812’, ‘猫’)]	[(‘86436’, ‘猫’)]
4	4	40242	[0.0,1.0]	[0.6,1.0]	规则_1	[‘407809’]	[(‘40242’, ‘407809’)]		
4	4	757646	[0.0,1.0]	[1.0,1.0]	规则_2	[‘423053’, ‘548848’]	[(‘757646’, ‘423053’), (‘757646’, ‘548848’)]	[(‘548848’, ‘猫’)]	[(‘757646’, ‘猫’)]
5	5	420093	[0.0,1.0]	[0.6,1.0]	规则_1	[‘275269’, ‘472129’]	[(‘420093’, ‘275269’), (‘420093’, ‘472129’)]		
5	5	1334826	[0.0,1.0]	[1.0,1.0]	规则_2	[‘1486432’]	[(‘1334826’, ‘1486432’)]	[(‘1486432’, ‘鱼’)]	[(‘1334826’, ‘鱼’)]
5	5	196947	[0.0,1.0]	[0.6,1.0]	规则 1_	[‘212129’]	[(‘196947’, ‘212129’)]		
6	6	348252	[0.0,1.0]	[1.0,1.0]	规则 2_	[‘1123497’]	[(‘348252’, ‘1123497’)]	[(‘1123497’, ‘猫’)]	[(‘348252’, ‘猫’)]
6	6	1144981	[0.0,1.0]	[0.6,1.0]	规则 1_	[‘232110’]	[(‘1144981’, ‘232110’)]		
6	6	420093	[0.0,1.0]	[1.0,1.0]	规则_2	[‘275269’, ‘472129’, ‘275337’]	[(‘420093’, ‘275269’), (‘420093’, ‘472129’), (‘420093’, ‘275337’)]	[(‘275337’, ‘海龟’)]	[(‘420093’, ‘海龟’)]
7	7	354365	[0.0,1.0]	[1.0,1.0]	规则 2_	[‘354455’, ‘718503’]	[(‘354365’, ‘354455’), (‘354365’, ‘718503’)]	[(‘718503’, ‘猫’)]	[(‘354365’, ‘猫’)]
7	7	420093	[0.0,1.0]	[1.0,1.0]	规则_2	[‘275269’, ‘472129’, ‘275337’]	[(‘420093’, ‘275269’), (‘420093’, ‘472129’), (‘420093’, ‘275337’)]	[(‘275337’, ‘海龟’)]	[(‘420093’, ‘海龟’)]
7	7	757646	[0.0,1.0]	[1.0,1.0]	规则_2	[‘423053’, ‘548848’]	[(‘757646’, ‘423053’), (‘757646’, ‘548848’)]	[(‘548848’, ‘猫’)]	[(‘757646’, ‘猫’)]
7	7	50219	[0.0,1.0]	[1.0,1.0]	规则 2_	[‘2067’, ‘50136’]	[(‘50219’, ‘2067’), (‘50219’, ‘50136’)]	[(‘2067’, ‘猫’), (‘50136’, ‘猫’)]	[(‘50219’, ‘猫’), (‘50219’, ‘猫’)]
7	7	148995	[0.0,1.0]	[0.6,1.0]	规则_1	[‘140490’]	[(‘148995’, ‘140490’)]		

算法 4更新解释

过程 update_interp(i', i)
    $updated \leftarrow 0$
    如果 $i(Lower) != i'(Lower)$ (or) $i(Upper) != i'(Upper)$ **则**
        $i(Lower) \leftarrow f_l(i(Lower)\ i'(Lower))$
                ▷ 默认情况下 f_l 是 $max()$ 函数，但也可以由用户定义。
        $i(Upper) \leftarrow f_u(i(Upper)\ i'(Upper))$
                ▷ 默认情况下， f_u 是 $min()$ 函数，但也可以由用户自行定义。
        $updated \leftarrow 1$
    结束 if
    如果 $updated$ (and) $(i,\ ic) \in$ **IPL**, $\forall ic$ **则**
        $ic(Lower) \leftarrow f_l(ic(Lower)\ 1 - i(Upper))$
        $ic(Upper) \leftarrow f_u(ic(Upper)\ 1 - i(Lower))$
    结束 if
    返回 $updated$
结束过程

算法 5一致性检查

过程 检查_一致性(i', i)
        ▷ i' 是带有 $[L', U']$ 的新值解释，并且， i 是带有 $[L, U]$ 的当前解释
    如果 $L' > U\,(or)\ U' < L$ **则**
        返回 $False$
    else
        返回 $True$
    结束 if
结束过程

算法 6不一致性消解

过程消解_不一致性($i \in \boldsymbol{I}$)
    $i(Lower) \leftarrow 0$
    $i(Upper) \leftarrow 1$
    $i\,(Static) \leftarrow 1$
结束过程