通过抽象解释实现最优程序合成 n

STEPHEN MELL，美国宾夕法尼亚大学STEVE ZDANCEWIC，美国宾夕法尼亚大学OSBERT BASTANI，美国宾夕法尼亚大学

我们研究如何合成带有数值常量的程序，使其在一组输入输出示例上优化某个定量目标，例如准确率。我们提出了一个通用框架，用于在给定的领域特定语言（DSL）中对这类程序进行最优合成，并提供可证明的最优性保证。该框架在通用搜索图中枚举程序，其中节点表示具体程序的子集。为提高可扩展性，该框架使用 A^* 搜索，并结合基于抽象解释的搜索启发式；直观而言，这种启发式为搜索图中的子树建立取值上界，使综合器能够识别并剪除可证明非最优的子树。此外，我们提出了一种自然策略，用于为单调语义构造抽象变换器；单调语义是数据分类领域特定语言中的组件常见的一种性质。最后，我们在两个现有的此类领域特定语言中实现了我们的方法，证明我们的算法比现有的最优综合器更具可扩展性。

CCS概念： • **计算理论 → 抽象**；**程序分析**。

A附加关键词与短语：程序合成、最优综合、抽象解释

ACM参考格式：
Stephen Mell、Steve Zdancewic 和 Osbert Bastani。2024年。通过抽象解释实现最优程序合成。*ACM*编程语言学报8，POPL，第16篇文章（2024年1月），25 页。https://doi.org/10.1145/3632858

arXiv:2602.14717v1 [cs.PL] 16 Feb 2026

1 引言

由于具有可解释性、鲁棒性和数据效率，近年来人们开始关注通过合成程序来解决数据处理和查询任务，包括处理图像和自然语言文本等半结构化和非结构化数据。示例包括 神经符号程序，它们结合深度神经网络（DNN）组件，从原始数据中提取语义信息 [Chen 等, 2021；Shah 等 2020]，，以及模糊匹配程序，它们使用具有定量语义的谓词来近似匹配实值数据 [Mell 等 2023]。例如，Shah 等 [2020] 合成用于标记序列数据的程序，Mell 等 [2023]合成对目标跟踪器输出的轨迹进行查询的程序，而Chen 等 [2021] 合成网页问答程序。大多数研究关注示例驱动编程（PBE），用户提供一组输入输出（IO）示例，目标是合成一个能够为每个输入生成正确输出的程序。此类程序的合成与传统示例驱动编程有两个关键区别：

- **Quantitative objectives:** The goal in neurosymbolic synthesis is typically to optimize a quantitative objective such as accuracy or F_1 score rather than to identify a program that is correct on all examples (which may be impossible).

作者地址： Stephen Mell，美国宾夕法尼亚大学，sm1@cis.upenn.edu； Steve Zdancewic，美国宾夕法尼亚大学，stevez@cis.upenn.edu； Osbert Bastani，美国宾夕法尼亚大学，obastani@seas.upenn.edu。

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2024 Copyright held by the owner/author(s).
ACM 2475-1421/2024/1-ART16
https://doi.org/10.1145/3632858

- **数值常量：** 处理模糊现实世界数据或深度神经网络组件输出的程序通常包含用作阈值的实值常数；例如，在查询视频轨迹时，一个常量可能是物体最大速度的阈值。

尽管这些性质偶尔会出现在传统示例驱动编程的场景中（例如，最小化资源消耗），但它们是神经符号综合中的根本问题。此外，在神经符号场景中，通常还存在可加以利用的额外结构，以提升综合性能——例如，某些数值分量可能关于其输入具有单调性。

大多数现有系统都专注于在特定领域专用语言（DSL）中合成示例。在这些场景中，先前的工作利用语义的单调性来剪枝搜索空间 [Chen 等，2021；Mell 等，2023]。一个通用框架是Shah 等，[2020], 该框架使用神经松弛来引导在通用 DSL 上进行搜索。从整体上看，他们使用 A^* 搜索来枚举 DSL 中的部分程序，这些程序以有向无环图（DAG）的形式表示。一般而言，A^* 搜索会根据评分函数（称为启发式）决定枚举部分程序的顺序，该函数会为每个部分程序映射出一个实值分数。当启发式是可采纳的——也就是说，对于该部分程序的任何补全，其输出都是目标值的上界（假设目标是最大化目标值）——那么 A^* 只要搜索终止，就能保证找到最优程序。[1]

Shah 等人 [2020] 提出了以下启发式方法：用未经训练的深度神经网络填充部分程序中的每个洞，然后以深度神经网络参数为变量最大化定量目标，并将所得目标值用作启发式。然而，只有在通常不符合实际情况的假设下，才能保证该评分函数是可采纳的：(i) 神经松弛具有足够的表达能力，可以表示领域特定语言中的任意程序，这需要非常大的深度神经网络；(ii) 深度神经网络参数的最大化过程会收敛到全局最优解，而对于随机梯度下降（SGD）等常见策略，这一假设并不成立。此外，随机梯度下降无法处理不可微目标函数，其中包括准确率和 F_1 评分。

因此，一个自然而然的问题是：我们能否构造出实际可用且可保证为可采纳的启发式方法？在这项工作中，我们从 演绎引导合成中汲取灵感；该方法使用 SMT 求解器等自动推理技术 [Bornholt 等，2016；Gulwani 等，2011；Solar-Lezama 等 2006] 或抽象解释 [Cousot 和 Cousot 1977] 来从搜索空间中剪枝部分程序——也就是说，证明给定部分程序的任何补全都无法满足给定的输入输出示例。具体而言，我们提出使用抽象解释为定量目标的合成构造启发式方法。传统上，抽象解释可以通过将每个洞替换为一个抽象值来剪枝部分程序；该抽象值会对给定输入下该洞可能取到的所有具体值进行过近似。这样，如果抽象输出不包含相应的具体输出，那么该部分程序就不可能被补全为满足该输入输出示例的程序，因此可以将其剪枝。

Our key insight is that abstract interpretation can similarly be used to construct an admissible heuristic for a quantitative objective. Essentially, we can use abstract interpretation to overapproximate the possible objective values obtained by any completion of a given partial program; then, the supremum of concrete values represented by the abstract output serves as an upper bound of the objective, so it can be used as an admissible heuristic. Thus, given abstract transformers for the DSL components and for the quantitative objective, our framework can synthesize optimal programs.

In addition, we propose general strategies for constructing abstract domains and transformers for common DSLs and objectives. As discussed above, many DSLs have monotone components. In

[1]We mean optimal on the given IO examples. This notion ignores suboptimality due to generalization error, which can be handled using standard techniques from learning theory [Kearns and Vazirani 1994].

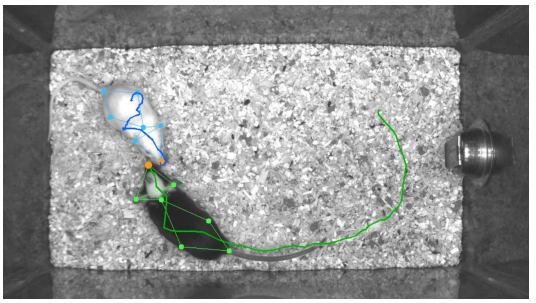


Fig. 1. A frame from a video of two mice interacting [Sun et al. 2021]; the mice are very close together, and are exhibiting the "sniff" behavior. The video has been processed using deep neural networks to produce certain keypoints, which are shown.

these settings, a natural choice of abstract domain is to use intervals for the real-valued constants; then, a natural abstract transformer is to evaluate the concrete semantics on the upper and lower bounds of the intervals. This strategy can straightforwardly be shown to correctly overapproximate the concrete semantics.

We implement our approach in the context of two DSLs—namely, the NEAR [Shah et al. 2020] DSL for the CRIM13 [Burgos-Artizzu et al. 2012] dataset, and the Quivr [Mell et al. 2023] DSL and benchmark. In our experiments, we demonstrate that our approach significantly outperforms an adaptation of Metasketches [Bornholt et al. 2016]—an existing optimal synthesis framework based on SMT solving—to our setting, as well as an ablation that uses breadth first search instead of A^* search. Our approach significantly outperforms both of these baselines in terms of running time. In summary, our contributions are:

- We propose a novel algorithm for optimal synthesis which performs enumerative search over a space of *generalized partial programs*. To prioritize search, it uses the A^* algorithm with a search heuristic based on abstract interpretation. If it returns a program, then that program is guaranteed to be optimal (Section 3).
- In practice, many DSLs have types that are equipped with a partial order, such as the real numbers or Booleans. For these types, we propose to use intervals as the abstract domains. For monotone DSL components—i.e., the concrete semantics respects the partial orders—a natural choice of abstract transformer is to simply apply the concrete semantics to the lower and upper bounds of the interval (Section 4).
- We implement our framework in the context of two existing DSLs (Section 5) and show in our experiments that it outperforms Metasketches [Bornholt et al. 2016], a state-of-the-art optimal synthesis technique based on SMT solvers, and a baseline that uses breadth-first search instead of our search heuristic (Section 6).

2 MOTIVATING EXAMPLE

我们考虑这样一个任务：目标是根据一段记录小鼠互动的视频，合成一个用于预测小鼠行为的程序 [Shah 等人 2020], 受到生物学数据分析问题的启发。具体而言，生物学家使用小鼠作为模式动物，以研究基本的生物过程并开发新的治疗干预措施；这有时需要确定某种干预对小鼠行为模式的影响，例如小鼠与其他小鼠互动的性质和持续时间。例如，图 1 展示了围栏中的两只小鼠，它们正在进行"嗅闻"行为。

进行这种行为分析通常需要研究人员观看数小时的视频，并手动标注其中的行为，这一过程十分耗费人力。因此，程序合成被应用于实现该任务的自动化 [Shah 等人 2020]的方法首先使用目标跟踪器，在视频的各帧中跟踪每只小鼠，从而生成由每只小鼠的一系列二维位置表示的轨迹。然后，他们为轨迹中的每一步提取特征——例如，如果视频中有两只小鼠，那么其中一个特征可能是它们在每一帧中的距离。基于这一系列特征，目标是预测小鼠在每一帧中（如果有的话）所进行行为的标签（生成一个标签序列，每一帧对应一个标签）。Shah 等人 [2020] 通过在用于处理轨迹的函数式领域专用语言（NEAR DSL）中合成程序来解决这一问题。总而言之，目标是合成一个以特征向量序列为输入、并输出标签序列的程序。我们考虑示例驱动编程（PBE）设置：给定若干由人类标注的示例，目标是合成一个能够最大化某个目标函数（例如准确率或 F_1 得分）的程序。

例如，考虑合成这样一个程序：给定一条特征化轨迹，它表示视频中的两只小鼠，程序会输出每个时间步小鼠的行为。具体而言，输入是一条轨迹 $x \in \mathbb{R}^*$（其中 T^* 表示由 T 组成的列表），其中时间步 t 上的特征 $x[t] \in \mathbb{R}$ 编码了该时间步两只小鼠之间的距离；输出为 $y \in \{\mathbf{f},\mathbf{t}\}^*$，其中$y[t]$ 编码了小鼠在时间步 t是否表现出“嗅闻”行为（$y = \mathbf{t}$ 表示是，$y = \mathbf{f}$ 表示否）。

我们考虑基于一个训练示例 $x_1 = (101,65)$ 和$y_1 = (\mathbf{f},\mathbf{t})$ 合成这样的程序（即，第一帧的鼠标距离为 101，标签为“未嗅闻”；第二帧的鼠标距离为 65，标签为“嗅闻”）。我们的目标是找到一个能够很好地对视频数据集进行分类的程序——也就是说，如果我们在这些视频上运行程序以获得预测标签，再计算预测标签与真实标签之间的分类指标（例如准确率），那么差异应当很小。给定程序 p，如果 $[\![p]\!](x_1) = y_1$，其准确率为 1；如果$[\![p]\!](x_1)[0] = y_1[0]$，但 $[\![p]\!](x_1)[1] \neq y_1[1]$（或反之），则准确率为 1/2；其他情况则为 0（其中 $x[i]$是序列 x中的第 i项）。

考虑候选程序“map($d \leq 50$)”。在传统语法中，该程序会写作“map ($\lambda d\,.\,d \leq 50$) x”，但这里的输入是单独指定的，并且省略了 λ（采用组合子风格，类似于 NEAR DSL）。该程序会对帧序列进行 map 操作，并针对每一帧输出该帧中的鼠标距离是否小于或等于 50。例如，在 x_1上求值时，它会输出

$$[\![\mathrm{map}(d \leq 50)]\!](x_1) = (\mathbf{f},\mathbf{f}).$$

因此，其准确率为 1/2，因为它正确标记了第一帧，但没有正确标记第二帧。

一种计算最优程序（即最大化目标函数的程序）的策略，是枚举 DSL 中的部分程序（即包含空位 、表示需要填入相应内容才能得到具体程序的程序），对每个具体程序计算目标函数，然后选出最佳程序。这种方法面临几个挑战：

- 与传统综合不同，在传统综合中，我们可以在找到满足给定规约的具体程序后停止枚举；而对于最优程序合成，我们需要枚举所有程序，否则就有可能返回一个非最优程序。

- 传统综合器会采用各种技术来剪枝搜索空间，从而提升可扩展性。例如，它们可能会通过演绎证明，部分程序的任何补全都无法满足规约——也就是说，无论如何填充部分程序中的洞，规约都不会成立。然而，这些技术并不能直接应用于最优合成。

- 合成实值常数会带来一个问题：一种方法是将常数离散化，枚举所有可能取值，再选出最佳值，但这种方法可能会慢得难以接受。例如，假设我们正在枚举以下部分程序的补全

$$\text{map}(d \le ??),$$

其中 ?? 是一个空位，需要填入一个实数值 $\theta \in [0, 100]$，才能得到一个具体程序。如果我们将 $\theta \in \{0, 1, \ldots\ldots, 100\}$ 离散化，就需要枚举 $\text{map}(d \le 0)$，……，$\text{map}(d \le 100)$，在 (x_1, y_1) 上分别对它们进行评估以衡量准确率，并选择准确率最高的程序。

我们的框架采用了两项关键创新来应对这些挑战：

- **广义部分程序：**我们的框架以传统的部分程序概念为基础，即用带孔语法的补全来表示一组具体程序，并将其扩展为更一般的程序集合，同时为其配备有向无环图（DAG）结构。这样，我们便能避免对实值常数进行离散化，并缩小搜索空间的分支因子。

- **A* 搜索：**我们的框架不再以任意顺序（例如广度优先搜索）枚举程序，而是使用 A^* 搜索来枚举程序。这样，我们就能避免考虑搜索空间中的所有程序，这与演绎剪枝类似，同时仍能返回最优程序。

下面我们将更详细地介绍这些技术。

广义部分程序。传统上，部分程序的搜索空间是一个有向无环图，其中节点是部分程序；如果通过使用 DSL 中的某个产生式填充 $\hat{}\,p$ 中的一个洞可以得到 $\hat{}\,p'$，那么二者之间就有一条边 $\hat{}\,p \to \hat{}\,p'$。例如，存在一条边

$$\text{map}(d \le ??) \to \text{map}(d \le 50)$$

在这个有向无环图中，因为我们用值 50 填充了这个洞。然而，即使我们将搜索空间离散化，填充这个洞也有 101 种方式。因此，如果 $\text{map}(d \le ??)$ 至少有一种有效的补全，我们就不能将其从搜索空间中剪枝（即使暂且不考虑我们想要的是最优程序，而不只是任意一个有效程序）。

相反，在我们的框架中，只要每个节点表示一组具体程序，且该节点的子节点共同表示这组程序，我们就允许搜索有向无环图包含的不仅仅是带洞的程序。作为这一通用框架的一种实用实现，我们考虑这样的部分程序：其中实数常量的洞可以标注约束，以限定可用于填充它们的值。例如，广义部分程序

$$\text{map}(d \le ??_{[50,100]})$$

represents the set of concrete programs

$$\{\text{map}(d \le \theta) \mid \theta \in [50, 100]\}$$

Then, the children of this generalized partial program in the search DAG should split the constraint in a way that covers the search space—e.g.,

$$\text{children}(\text{map}(d \le ??_{[50,100]})) = \{\text{map}(d \le ??_{[50,75]}), \text{map}(d \le ??_{[75,100]})\}.$$

这种策略为剪枝搜索空间提供了更多机会。例如，即使我们无法剪去程序映射 $\text{map}(d \le ??_{[50,100]})$，也可能可以剪去程序映射 $\text{map}(d \le ??_{[50,75]})$。这样，我们就不必枚举 51 个程序（即分别对应每个 $\theta \in \{50、\ldots、100\}$ 的程序），而只需剪去 $\text{map}(d \le ??_{[50,75]})$ 并评估 26 个程序（即分别对应每个 $\theta \in \{75、\ldots、100\}$ 的程序）。当然，我们还可以进一步细分搜索空间，从而进一步减少枚举量。

A^* 搜索。接下来，我们将介绍如何通过使用$A*$ 搜索，实现最优综合中与剪枝相对应的做法。从整体上看， A^* 搜索按照某种启发式枚举搜索图中的节点；就我们的目的而言，启发式是一个将部分程序 $\hat{p}$ 映射到实数 μ 的函数。如果某个启发式是任意补全 $\hat{p}$所能达到的最佳目标值的上界，则称其为可采纳——即，

$$p \in \text{completions}(\hat{p}) \Rightarrow \mu \geq \text{objective}(p).$$

The heuristic adapts deductive reasoning to optimal synthesis: whereas deductive reasoning guarantees that no completion of $\hat{p}$ can satisfy the given specification, the heuristic guarantees that no completion of $\hat{p}$ can achieve objective value greater than μ—e.g., if we find a concrete program with objective value $\geq \mu$, we can safely prune completions of $\hat{p}$ from the search DAG.

While A^* search has previously been used in synthesis for quantitative objectives [Shah et al. 2020], the heuristics used are not admissible and so do not provide theoretical guarantees. Our key contribution is showing that abstract interpretation can naturally be adapted to design admissible heuristics. Abstract interpretation can be used for traditional deductive reasoning as follows: fill each hole in the current partial program with an abstract value $\top$, evaluate the partial program using abstract semantics, and check if the abstract output is consistent with the specification.

In our example, a natural choice of abstract domain for constants is the interval domain. In addition, rather than fill each hole with $\top$, if a hole has a constraint $??_{[a,b]}$, we can instead fill it with the interval $[a, b]$. For instance, for our example program $\text{map}(d \leq ??_{[50,75]})$, we can fill the hole with the interval $[50, 75]$ to obtain $\text{map}(d \leq [50, 75])$. Then, the abstract semantics $[\![\cdot]\!]^\#$ evaluate as follows:

$$[\![\text{map}(d \leq ??_{[50,75]})]\!]^\#(x_1) = ([\![101 \leq ??_{[50,75]}]\!]^\#, [\![65 \leq ??_{[50,75]}]\!]^\#) = (\mathbf{f}, \top). \tag{1}$$

In other words, the first element is $\mathbf{f}$ since $x_1 = (101, 65)$, and we know $101 \not\leq \theta$ for any $\theta \in [50, 75]$, and the second element is $\top$ since the relationship between 65 and $\theta \in [50, 75]$ can be either $\mathbf{f}$ or $\mathbf{t}$. Importantly, here the abstract values are over holes in the program rather than over the input to it.

Traditionally, we would then check whether this abstract output is consistent with the specification. For optimal synthesis, we observe that we can define an abstract semantics for the objective function. In our example, we can compute an "abstract accuracy" as follows (where $\mathbb{1}$ is a indicator function, and $\leq$, $\mathbb{1}$, $+$, $\cdot$, and $=$ are abstracted in the obvious way):

$$\begin{aligned}
\text{accuracy}^\#(\hat{p}) &= \frac{1}{2}\sum_{t=1}^{2} \mathbb{1}\left[[\![\text{map}(d \leq ??_{[50,75]})]\!]^\#(x_1)[t] = y_1[t]\right] \\
&= \frac{1}{2}\left(\mathbb{1}[\mathbf{f} = \mathbf{f}] + \mathbb{1}[\top = \mathbf{t}]\right) \\
&= \frac{1}{2}([1, 1] + [0, 1]) \\
&= [1/2, 1].
\end{aligned} \tag{2}$$

换句话说，对于第一帧，$\text{map}(d \leq ??_{[50,75]})$ 表示的具体程序都预测为 $\mathbf{f}$，这与 $y_1[1]$ 相等，因此该帧总是会被正确分类。相比之下，对于第二帧，具体程序可能输出 $\mathbf{f}$ 或 $\mathbf{t}$，因此我们无法确定该帧是否被正确分类。所以，真实准确率位于区间 $[1/2, 1]$ 内。由于抽象解释保证会对语义进行过近似，我们可以将抽象目标值的上界用作启发式——例如，对于 $\text{map}(d \leq [50, 75])$，该启发式计算得出 $\mu = 1$。

最后，我们总结从搜索有向无环图中的节点 $\text{map}(d \leq ??_{[0,100]})$ 开始的完整搜索过程，其抽象目标值为 $[1/2, 1]$。首先，我们将其拆分为

$$\hat{p}_1 = \text{map}(d \leq ??_{[0,50]}) \qquad \text{and} \qquad \hat{p}_2 = \text{map}(d \leq ??_{[50,100]}).$$

The abstract accuracies of $\hat{p}_1$ and $\hat{p}_2$ are $[1/2, 1/2]$ and $[1/2, 1]$, so their heuristic values are $\mu_1 = 1/2$ and $\mu_2 = 1$, respectively. Since $\mu_2 > \mu_1$, our algorithm explores $\hat{p}_2$ next, splitting it into

$$\hat{p}_3 = \text{map}(d \leq ??_{[50,75]}) \qquad \text{and} \qquad \hat{p}_4 = \text{map}(d \leq ??_{[75,100]}),$$

which have abstract accuracies of $[1/2, 1]$ and $[1, 1]$, respectively, and heuristic values of $\mu_3 = 1$ and $\mu_4 = 1$, respectively. In this example, the lower bound on the accuracy of $\hat{p}_4$ is also 1, so we know that any choice $\theta \in [75, 100]$ for filling this hole is guaranteed to achieve an accuracy of 1; thus, any concrete program $\text{map}(d \leq \theta)$ such that $\theta \in [75, 100]$ is optimal, and our algorithm can terminate without ever considering $\hat{p}_1$ or $\hat{p}_3$.

In general, our algorithm terminates once the range of possible optimal values is sufficiently small. For each node on the search frontier, we have an upper and lower bound on the objective value of all the programs it represents. The greatest of these lower bounds provides a lower bound on the best possible objective value, and the greatest of these upper bounds provides an upper bound on the best possible objective value. As a result, once the difference between the bounds is $\leq \epsilon$, we know that we have a program within ϵ of being optimal.

3 OPTIMAL SYNTHESIS VIA ABSTRACT INTERPRETATION

In this section, we consider the program synthesis problem where: (i) programs in the domain-specific language may have real-valued constants, and (ii) the synthesis objective is real-valued, where the goal is to return the optimal program (Section 3.1). Then, we describe our synthesis algorithm for solving this problem, which uses A^* search in conjunction with a search heuristic based on abstract interpretation (Section 3.2).

3.1 Problem Formulation

Domain-Specific Language. For concreteness, consider a DSL whose syntax is given by a context-free grammar $\mathcal{G} = (V, \Sigma, R, P)$, where V is the set of nonterminals, Σ is the set of terminals, $P \in V$ is the start symbol, and R is the set of productions

$$P ::= X \mid c \mid f(P, ..., P) \qquad (c \in \mathcal{C}, f \in \mathcal{F})$$

where X is a symbol representing the input, $c \in \mathcal{C}$ is a constant (including real-valued constants, i.e., $\mathbb{R} \subseteq \mathcal{C}$), and $f \in \mathcal{F}$ is a DSL component (i.e., function), but our framework extends straightforwardly to more general grammars. We let $p \in \mathcal{P} = \mathcal{L}(\mathcal{G}) \subseteq \Sigma^*$ denote the concrete programs in our DSL. Furthermore, we assume the DSL has denotational semantics $[\![\cdot]\!]$, where $[\![p]\!] : \mathcal{X} \to \mathcal{Y}$ maps inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$ according to the following rules:

$$[\![X]\!](x) = x, \quad [\![c]\!](x) = c, \quad [\![f(p_1, ..., p_k)]\!](x) = f([\![p_1]\!](x), ..., [\![p_k]\!](x)),$$

where we assume the functions $f : \mathcal{X}_1 \times ... \times \mathcal{X}_k \to \mathcal{Y}$ are given.

In Section 2, we considered programs like $\text{map}(d \leq 50)$, which we will use as a running example in this section. They are simplified versions of programs from the NEAR DSL, and are generated by the grammar (where d represents the input)

$$E ::= d \mid c \mid \text{map}(E) \mid E \leq E \qquad (c \in \mathbb{R})$$

Task Specification. We consider programming by example (PBE), where each task is specified by a sequence $Z \in \mathcal{Z} := (\mathcal{X} \times \mathcal{Y})^*$ of input-output (IO) examples, and the goal is to compute a program p^* that maximizes a given quantitative objective $\phi : \mathcal{P} \times \mathcal{Z} \to \mathbb{R}$

$$p^* \in \arg\max_{p \in \mathcal{P}} \phi(p, Z).$$

ϕ是将语义应用于示例（x, y）的函数 $\in$ Z——即存在一个函数ϕ_0：$(\mathrm{Y} \times \mathrm{Y})^* \to \mathrm{R}$，使得

$$\phi(p, Z) = \phi_0\left(\{(\llbracket p \rrbracket(x), y)\}_{(x,y)\in Z}\right), \tag{3}$$

在我们的运行示例中，我们按如下方式选择 ϕ_0，因此 ϕ（p, Z）是 P在 Z上的准确率：

$$\phi_0(W) = \frac{1}{|W|} \sum_{(y', y) \in W} \mathbb{1}[y' = y],$$

其中 $\mathbb{1}[\cdot]$ 是指示函数。

部分程序。 基于示例的编程的一种常见策略是枚举部分程序——即领域特定语言中带有空位的程序——以尝试找到满足给定输入输出示例的具体程序。直观而言，部分程序是文法 G 中的部分推导。为形式化这一概念，给定两个序列 $\hat{}\, p, \hat{}\, p' \in (\Sigma \cup V)^*$，若可通过用该非终结符的产生式 $r = N_i \to M_1 ... M_k \in R$的右部替换 $\hat{}\, p$ 中的非终结符 $N_i \in V$而得到 $\hat{}\, p'$，则我们记作 $\hat{}\, p \to \hat{}\, p'$——即 $\hat{}\, p = N_1 ... N_i ... N_h$与 $\hat{}\, p' = N_1 ... M_1 ... M_k ... N_h$。我们将此关系记为 $\hat{}\, p' = \mathrm{fill}(\hat{}\, p, i, r)$——即通过使用产生式 r填充 $\hat{}\, p$中的第 i 个洞 N_i得到 $\hat{}\, p'$。接下来，若存在序列 $\hat{}\, p = \hat{}\, p_1 \to ... \to \hat{}\, p_n = \hat{}\, p'$，则我们记 $\hat{}\, p \xrightarrow{*} \hat{}\, p'$，并称 $\hat{}\, p'$ 可以推导 自 $\hat{}\, p$。

Note that concrete programs $p \in \mathcal{P}$ are sequences $p \in \Sigma^*$ that can be derived from the start symbol P (i.e. $P \xrightarrow{*} p$). Similarly, a partial program is a sequence $\hat{p} \in \hat{\mathcal{P}} := (\Sigma \cup V)^*$ that can be derived from P—i.e. $P \xrightarrow{*} \hat{p}$. The only difference is that $\hat{p}$ may contain nonterminals, which are called *holes*. The space of partial programs naturally forms a directed acyclic graph (DAG) via the relation $\hat{p} \to \hat{p}'$; note that concrete programs are leaf nodes in this DAG. Thus, we can perform synthesis by enumerating partial programs according to the structure of this DAG. Furthermore, given a partial program $\hat{p} \in \hat{\mathcal{P}}$ and a concrete program $p \in \mathcal{P}$, we say p is a *completion* of $\hat{p}$ if $\hat{p} \xrightarrow{*} p$ (i.e., p can be obtained from $\hat{p}$ by iteratively filling the holes of $\hat{p}$).

In our running example, traditional partial programs correspond to the grammar

$$\bar{E} ::= ?? \mid d \mid c \mid \mathrm{map}(\bar{E}) \mid \bar{E} \le \bar{E} \qquad (c \in \mathbb{R}).$$

Generalized Partial Programs. A key challenge is searching over real-valued constants $c \in \mathbb{R}$. Our grammar contains infinitely many productions of the form $P \to \theta$ for $\theta \in \mathbb{R}$, and even if we discretize this search space, the number of productions is still large in practice.

To address this challenge, we propose a strategy where we enumerate *generalized partial programs* $\hat{p} \in \hat{\mathcal{P}}$, which generalize (i) the fact that partial programs correspond to sets of concrete programs (i.e. the set of their completions), and (ii) the DAG structure of partial programs.

Definition 3.1. A space of *generalized partial programs* is a set $\hat{\mathcal{P}}$ together with a *concretization function* $\gamma : \hat{\mathcal{P}} \to 2^{\mathcal{P}}$ and a *DAG structure* $\mathrm{children} : \hat{\mathcal{P}} \to 2^{\hat{\mathcal{P}}}$, such that

$$\gamma(\hat{p}) = \bigcup_{\hat{p}' \in \mathrm{children}(\hat{p})} \gamma(\hat{p}'). \tag{4}$$

直观地说，$\gamma(\hat{p}) \subseteq \mathrm{P}$ 是由抽象程序 $\hat{p}$ 所表示的具体程序集合。此外，子节点在 $\hat{\mathrm{P}}$ 上编码了一种与 γ 兼容的有向无环图结构——即 $\hat{p}$ 的子节点 $\hat{p}'$ 必须共同包含 $\hat{p}$ 中的所有具体程序。

For example, to capture traditional partial programs, we let

$$\gamma(\hat{p}) = \{p \in \mathcal{P} \mid \hat{p} \xrightarrow{*} p\},$$

i.e. p is a completion of $\hat{p}$, and

$$\mathrm{children}(\hat{p}) = \{\hat{p}' \in \hat{\mathcal{P}} \mid \hat{p} \to \hat{p}'\}.$$

In Section 4, we will propose generalized partial programs that can include constraints on real-valued holes—e.g., $??_{[0,1]}$ is a partial program that can only be filled by a real value $\theta \in [0, 1]$.

Finally, a simple way to satisfy Definition 3.1 is to define γ based on the children function—i.e., we can define $p \in \gamma(\hat{p})$ if there exists a sequence $\hat{p} = \hat{p}_1, \ldots, \hat{p}_n = p$ of generalized partial programs such that $\hat{p}_{j+1} \in \mathrm{children}(\hat{p}_j)$ for all $1 < j < n$. In other words, we can reach the concrete program p from the generalized partial program $\hat{p}$ in the search DAG. This strategy straightforwardly guarantees (4), since by definition, every $p \in \gamma(\hat{p})$ must be the descendant of some child of $\hat{p}$.

在我们的运行示例中，广义部分程序对应于文法

$$\hat{E} ::= ??_{[a,b]} \mid ?? \mid d \mid c \mid \mathrm{map}(\hat{E}) \mid \hat{E} \le \hat{E} \qquad (a, b \in \mathbb{R}),$$

而具体化函数满足

$$\gamma(\mathrm{map}(??)) = \{\mathrm{map}(p) \mid p \in \mathcal{P}\}$$
$$\gamma(\mathrm{map}(d \le ??_{[50,100]})) = \{\mathrm{map}(d \le c) \mid a \le c \le b\}$$

子节点函数满足

$$\mathrm{children}(\mathrm{map}(??)) = \{\mathrm{map}(d),\ \mathrm{map}(c),\ \mathrm{map}(\mathrm{map}(??)),\ \mathrm{map}(?? \le ??)\}$$
$$\mathrm{children}(\mathrm{map}(d \le ??_{[50,100]})) = \{\mathrm{map}(d \le ??_{[50,75]}),\ \mathrm{map}(d \le ??_{[75,100]})\}。$$

抽象目标。目前，我们考虑一个抽象目标 $[\![\cdot]\!]^{\#}_{\phi}$，它直接将广义部分程序映射到抽象实数值；通常通过为每个组件 $f \in \mathrm{F}$ 以及目标函数 ϕ_0提供抽象变换器，并将它们组合起来，以组合方式构造。具体而言，抽象目标的类型为 $[\![\hat{p}]\!]^{\#}_{\phi} : \mathrm{Z} \to \hat{\mathrm{R}}$，其中 $\hat{\mathrm{R}}$ 是实数的抽象域，表示潜在的目标值 $\phi(p, Z)$（例如，区间 $\hat{}$ 域）。我们不需要为 R 指定具体化函数，而只需为这个 $\hat{}$ 抽象域指定一个上界——即一个函数 $\mu : \mathrm{R} \to \mathrm{R}$，它表达了这样的直觉："$\mu(\hat{r})$ 大于包含在 $\hat{r}$ 中的任意实数 r"。

Definition 3.2. Given objective ϕ and generalized partial programs $(\hat{\mathcal{P}}\ \gamma, \mathrm{children})$, an abstract objective $([\![\cdot]\!]^{\#}_{\phi}, \mu)$ is *valid* if

$$\big(p \in \gamma(\hat{p})\big) \Rightarrow \big(\mu([\![\hat{p}]\!]^{\#}_{\phi}(Z)) \ge \phi(p, Z)\big) \qquad (\forall p \in \mathcal{P}), \tag{5}$$

and

$$\mu([\![p]\!]^{\#}_{\phi}(Z)) = \phi(p, Z) \qquad (\forall p \in \mathcal{P}). \tag{6}$$

Intuitively, (5) says that $\mu([\![\hat{p}]\!]^{\#}_{\phi}(Z))$ is an upper bound on the objective value $\phi(p, Z)$ for concrete programs p is contained in the abstract program $\hat{p}$. In addition, (6) says that for concrete programs, the abstract objective and concrete objective coincide.

$\hat{}$ $\hat{}$最后，$\mathbb{R}$的一个典型选择是区间空间 $\mathbb{R}$，其中 $(r, r') \in \mathbb{R}$ 表示实数集合 $\{r'' \in \mathbb{R} \mid r \le r'' \le r'\}$（详情请参见定义 4.1）。那么，上界为

算法 1 我们的算法以任务规格 Z 为输入，同时还接收一个领域特定语言 G、一个广义程序空间 ($\hat{}$P γ)、抽象目标函数 ($[\![\cdot]\!]^{\#}_{\phi}, \mu$)、目标下界 ν 和目标误差容限 ε，并返回任务 Z 的最优程序 p^*。为此，它在 A^* 搜索中将抽象目标函数用作启发式，从初始广义部分程序 $\hat{} p_0$ 开始搜索。

procedure A^*-Synthesis($Z, \mathcal{G}, \hat{\mathcal{P}}, \gamma, [\![\cdot]\!]^{\#}_{\phi}, \mu$)
  $h \leftarrow$ heap(sort_by = $\lambda \hat{p}.\mu([\![\hat{p}]\!]^{\#}_{\phi}(Z))$)
  h.push($\hat{p}_0$)
  while true do
    $\hat{p} \leftarrow h$.pop_greatest()
    if $\max\{\mu([\![\hat{p}']\!]^{\#}_{\phi}(Z)) \mid \hat{p}' \in h\} - \max\{\nu([\![\hat{p}']\!]^{\#}_{\phi}(Z)) \mid \hat{p}' \in h\} \leq \varepsilon$ **then**
      return $\arg\max\{\nu([\![\hat{p}']\!]^{\#}_{\phi}(Z)) \mid \hat{p}' \in h\}$
    end if
    for $\hat{p}' \in$ children($\hat{p}$) **do**
      h.push($\hat{p}'$)
    end for
  end while
end procedure

由 $\mu((r, r_\prime)) = r_\prime$。抽象目标 $[\![\cdot]\!]^{\#}_{\phi}$取决于 DSL；我们将在第4节介绍一种通用构造。

在我们的运行示例中，目标函数 ϕ 根据方程 3 分解为语义$[\![\cdot]\!]$ 和准确率 ϕ_0。因此，我们根据抽象语义 $[\![\cdot]\!]^{\#}_{\phi}$ 来定义抽象目标$[\![\cdot]\!]^{\#} : \hat{}\mathcal{P} \rightarrow \hat{}\mathbb{R}$ (方程 1) 和抽象准确率（方程2）。

3.2 A^* 通过抽象解释进行合成

给定一组输入输出示例，算法 1 使用 A^* 搜索广义部分程序 $\hat{} p \in \hat{}$P，并结合启发式 $\hat{} p \mapsto \mu([\![\hat{} p]\!]^{\#}_{\phi}(Z))$，以计算最优程序。具体而言，它使用堆 h 跟踪搜索有向无环图前沿中的广义部分程序 $\hat{} p$；在每次迭代中，弹出当前最佳节点 $\hat{} p$，然后枚举 $\hat{} p$ 的子节点 $\hat{} p'$，并根据启发式将它们添加到 h。当 $\hat{} p \in h$ 上的 $\hat{}$ 目标下界 ν: R $\rightarrow$ R（与 μ 类似，但它是抽象目标的下界而非上界）与 $\hat{} p \in h$ 上界 μ 的最大值之差在容差 $\varepsilon \geq 0$ 范围内时，算法终止，并返回下界最高的广义部分程序。当目标函数的抽象 R 为实数区间时，$\nu((r, r')) = r$是一个自然的选择。

For our running example, consider the following generalized partial programs and their abstract objective values:

$$\hat{p}_0 \coloneqq \text{map}(d \leq ??_{[0,100]}) \qquad [\![\hat{p}_0]\!]^{\#}_{\phi} = [1/2, 1]$$
$$\hat{p}_1 \coloneqq \text{map}(d \leq ??_{[0,50]}) \qquad [\![\hat{p}_1]\!]^{\#}_{\phi} = [1/2, 1/2]$$
$$\hat{p}_2 \coloneqq \text{map}(d \leq ??_{[50,100]}) \qquad [\![\hat{p}_2]\!]^{\#}_{\phi} = [1/2, 1]$$
$$\hat{p}_3 \coloneqq \text{map}(d \leq ??_{[50,75]}) \qquad [\![\hat{p}_3]\!]^{\#}_{\phi} = [1/2, 1]$$
$$\hat{p}_4 \coloneqq \text{map}(d \leq ??_{[75,100]}) \qquad [\![\hat{p}_4]\!]^{\#}_{\phi} = [1, 1]$$

The search process starts with $h_0 = \{(\hat{p}_0 : 1)\}$. In the first iteration, $\hat{p}_0$ is popped, since it (trivially) has the highest heuristic value. Its children, $\hat{p}_1$ and $\hat{p}_2$ are pushed, resulting in $h_1 = \{\hat{p}_2 : 1, \hat{p}_1 : 1/2\}$.

接下来，我们弹出 $\hat{}\,p_2$，因为它的启发式值最高。在这里，我们可以看到对A^* 进行的“软剪枝”发挥了作用：我们知道，$\hat{}\,p_1$ 的后代无法达到 $> 1/2$ 的目标值；而就我们目前所知，$\hat{}\,p_2$ 的后代可能达到高达 1 的目标值。我们不能直接剪去 $\hat{}\,p_1$ 本身，因为最优目标值可能是 $\leq 1/2$；不过，我们可以暂缓考虑 $\hat{}\,p_1$的后代，直到确定最优目标值为 $\leq 1/2$。接着，我们将 $\hat{}\,p_2$的子节点 $\hat{}\,p_3$ 和 $\hat{}\,p_4$ 推入堆中，得到 $h_2 = \{\hat{}\,p_3, : 1,\ \hat{}\,p_4 : 1,\ \hat{}\,p_1 : 1/2\}$。

最后，我们注意到，对于堆中的抽象目标区间（$\{[1/2, 1], [1, 1], [1/2,\ 1/2]\}$），最大下界（1）与最小上界（1）之间的距离为 0，因此搜索终止，并返回 $\hat{}\,p_4$。

Let $\phi^*_Z \coloneqq \max_{p' \in \mathcal{P}} \phi(p', Z)$ be the optimal objective value. We have the following optimality guarantee:

Theorem 3.3. *If Algorithm 1 returns an abstract program $\hat{p}$, then $\hat{p}$ is ϵ-optimal—i.e. $\forall p \in \gamma(\hat{p})$, $\phi^*_Z - \phi(p, Z) \leq \epsilon$.*

Proof. By (4), Algorithm 1 preserves the invariant that every program $p \in \mathcal{P}$ is contained in some generalized partial program $\hat{p} \in h$—i.e.,

$$\bigcup_{\hat{p} \in h} \gamma(\hat{p}) = \mathcal{P}.$$

This property can be checked by an easy induction argument on the while loop iteration. This implies that

$$\phi^*_Z = \max\{\max\{\phi(p', Z) \mid p' \in \gamma(\hat{p}')\} \mid \hat{p}' \in h\}$$

and since μ must overapproximate the concrete objective values,

$$\phi^*_Z \leq \max\{\mu([\![\hat{p}']\!]^\#_\phi(Z)) \mid \hat{p}' \in h\}.$$

Next, suppose Algorithm 1 terminates, returning $\hat{p} \in \hat{\mathcal{P}}$, and let p be any concrete program in $\gamma(\hat{p})$. The termination condition ensures that

$$\phi^*_Z \leq \nu([\![\hat{p}]\!]^\#_\phi(Z)) + \epsilon.$$

Finally, since ν must underapproximate the concrete objective values,

$$\phi^*_Z \leq \phi(p, Z) + \epsilon.$$

□

通过将有限有向无环图用作搜索空间，可以轻松确保终止；例如，我们可以对实值常量进行离散化来实现这一点。一般而言，只要每条无限链的抽象损失都收敛，我们就能保证收敛。例如，当目标函数关于实值程序参数是利普希茨连续时，这一性质便成立。（目标值的上界与下界之差不超过利普希茨常数乘以区间盒的直径；随着搜索进行、区间盒不断缩小，该差值也趋于零。）然而，利普希茨连续性往往并不成立；我们将确保收敛的其他方法留待未来工作探索。

4 INSTANTIATION FOR INTERVAL DOMAINS

第 3 节介绍了在已知目标领域特定语言的抽象语义和搜索有向无环图时进行最优合成的一般框架。本节将介绍一种自然策略，用于在领域特定语言的类型具有偏序时构造抽象语义。我们首先说明如何为偏序类型构造抽象域（第 4.1 节），如何为具有单调语义的各个组件构造抽象语义（第 4.2 节），以及如何为单调目标函数构造抽象变换器（第 4.3 节）。接着，我们介绍一种部分程序空间，其中对应实值常量的洞可以选择性地标注区间约束（第 4.4 节）。最后，我们展示如何结合抽象变换器和目标函数，对区间约束部分程序进行抽象解释（第 4.5节）。

4.1 Interval Domains from Partial Orders

Many DSLs have the property that their types are equipped with a partial order—e.g. the usual order on the real numbers, or the order $\mathbf{f} \leq \mathbf{t}$ on the Booleans.

Definition 4.1. Given a partially ordered set $\mathcal{Z}$, let $\bar{\mathcal{Z}} = \mathcal{Z} \cup \{-\infty, +\infty\}$, where $-\infty \leq z \leq +\infty$ for all $z \in \mathcal{Z}$. Then, the *interval domain* is the set $\hat{\mathcal{Z}} = \{(a, b) \in \bar{\mathcal{Z}}^2 \mid a \leq b\} \cup \{\bot\}$, together with an abstraction function $\alpha : \mathcal{Z} \to \hat{\mathcal{Z}}$ defined by $\alpha(z) = (z, z)$, and a concretization function $\gamma : \hat{\mathcal{Z}} \to 2^{\mathcal{Z}}$ defined by

$$\gamma((z_0, z_1)) = \{z \in \mathcal{Z} \mid z_0 \leq z \leq z_1\}.$$

In other words, γ maps (z_0, z_1) to the interval $[z_0, z_1]$. It is clear that $z \in \gamma(\alpha(z))$.

4.2 Interval Transformers for Monotone Functions

Next, we define our notion of abstract transformer for the interval domain. DSLs often contain functions that are monotone, respecting the partial orders between their input and output types.

Definition 4.2. Consider a function $f : \mathcal{X}_1 \times ... \times \mathcal{X}_k \to \mathcal{Y}$, and where $\mathcal{X}_1, ..., \mathcal{X}_k$ and $\mathcal{Y}$ are partially ordered sets. We say f is *monotone* if

$$\bigwedge_{i=1}^{k} x_i \leq x_i' \Rightarrow f(x_1, ..., x_k) \leq f(x_1', ..., x_k').$$

For example, the + operator is monotone in both of its inputs.

Definition 4.3. Given a monotone function $f : \mathcal{X}_1 \times ... \times \mathcal{X}_k \to \mathcal{Y}$, let $\hat{\mathcal{X}}_i$ be the interval domain for each $\mathcal{X}_i$ and $\hat{\mathcal{Y}}$ be the interval domain for $\mathcal{Y}$. The *interval transformer* for f is the function $\hat{f} : \hat{\mathcal{X}}_1 \times ... \times \hat{\mathcal{X}}_k \to \hat{\mathcal{Y}}$ defined by

$$\hat{f}((x_{1,0}, x_{1,1}), ..., (x_{k,0}, x_{k,1})) = (f(x_{1,0}, ..., x_{k,0}), f(x_{k,1}, ..., x_{k,1})).$$

If $x_{i,0} = -\infty$ for any $i \in [k]$, then we let the lower bound be $f(x_{1,0}, ..., x_{k,0}) = -\infty$, and if $x_{i,1} = +\infty$ for any $i \in [k]$, then we let the upper bound be $f(x_{k,1}, ..., x_{k,1}) = +\infty$.[2]

For example, since $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is monotone, then $\hat{+} : \hat{\mathbb{R}} \times \hat{\mathbb{R}} \to \hat{\mathbb{R}}$ is given by

$$(a, b) \hat{+} (c, d) \coloneqq (a + c, b + d)$$

The following key result shows that $\hat{f}$ overapproximates the concrete semantics:

[2]这里的“单调”指单调递增。对于单调递减的函数，我们可以交换区间变换器输出的端点。

ˆ引理 4.4. 设 $f: \mathcal{X}_1 \times ... \times \mathcal{X}_k \to \mathcal{Y}$ 是单调的，且令 f为其区间变换器。对于任意ˆ$x_i \in \mathcal{X}_i$和 ˆ$x_i = (x_{i,0}, x_{i,1}) \in \mathcal{X}_i$，若满足 $x_i \in \gamma(\hat{x}_i)$对所有 $i \in [k]$均成立，则有

$$f(x_1, ..., x_k) \in \gamma\left(\hat{f}(\hat{x}_1, ..., \hat{x}_k)\right).$$

Proof. For all $i \in [k]$, by our assumption that $x_i \in \gamma(\hat{x}_i)$, we have $x_{i,0} \leq x_i \leq x_{i,1}$. Thus, by monotonicity of f, we have

$$f(x_{1,0}, ..., x_{k,0}) \leq f(x_1, ..., x_k) \leq f(x_{1,1}, ..., x_{k,1}). \tag{7}$$

By definition of $\hat{f}$, we have $\hat{f}(\hat{x}_1, ..., \hat{x}_k) = (f(x_{1,0}, ..., x_{k,0})\ f(x_{1,1}, ..., x_{k,1}))$, so by (7) and definition of γ, we have $f(x_1, ..., x_k) \in \gamma(\hat{f}(\hat{x}_1, ..., \hat{x}_k))$, as claimed. □

In other words, if x_i is contained in the interval $\hat{x}_i$ for each i, then $f(x_1, ..., x_k)$ is contained in the interval $\hat{f}(\hat{x}_1, ..., \hat{x}_k)$. Thus, $\hat{f}$ overapproximates the concrete semantics of f.

4.3 单调目标函数的区间变换器

类似地，如果 ϕ_0是单调的——即对于 $W_i = \{(y'_{i,1}, y_{i,1}) ..., (y'_{i,k}, y_{i,k})\}$，若 $y'_{0,j} \leq y'_{1,j}$对所有 $j \in [k]$都成立，则有

$$\phi_0(W_0) \leq \phi_0(W_1).$$

请注意，我们只要求候选程序 p输出的标签$y'_{j,i}$具有单调性，而不要求真实标签 $y_{j,i}$具有单调性，因为后者始终是具体值。于是，我们可以构造抽象变换器 $\phi_0 : (\mathrm{Y} \times \mathrm{Y})^* \to \mathrm{R}$，如下所示

$$\hat{\phi}_0\left(\{((y'_{0,i}, y'_{1,i}), y_i)\}_{i=1}^k\right) = \left(\phi_0\left(\{(y'_{0,i}, y_i)\}_{i=1}^k\right), \phi_0\left(\{(y'_{1,i}, y_i)\}_{i=1}^k\right)\right).$$

4.4 带区间约束的部分程序

接下来，我们将介绍一类广义部分程序，它扩展了带有空洞注释的部分程序，这些注释用于约束可用于填充这些空洞的值。

ˆ定义 *4.5*。假设常量空间 C 是偏序集，并令 C 为其区间域。那么，区间约束部分程序 $\tilde{p} = (\hat{p}, \kappa)$是一个部分程序 ˆ$p$，以及一个映射 $\kappa : \mathrm{holes}(\hat{p}) \to \hat{\mathrm{C}} \cup \{\varnothing\}$，其中 $\mathrm{holes}(\hat{p}) \subseteq \mathrm{N}$ 表示 ˆp 中空位的索引。

For example, suppose that $\hat{p} = N_1 ... N_i ... N_h$ is a partial program, where N_i is a nonterminal (and therefore a hole), so $i \in \mathrm{holes}(\hat{p})$. Intuitively, an annotation $\kappa(i) = \hat{c}$ imposes the constraint that the value used to fill N_i must be a constant $c \in C$, and that c must satisfy $c \in \gamma(\hat{c})$ (i.e., c is contained in the interval $[c_0, c_1]$). Alternatively, if $\kappa(i) = \varnothing$, then no such constraint is imposed—i.e., N_i may be filled with any constant or a different production in the grammar. (Note that $\varnothing$ is not the same as providing the interval $[-\infty, +\infty]$ because that constraint would require this hole to be filled by a *constant*, whereas the optimal program might need a non-constant expression.) We let $\tilde{\mathcal{P}}$ denote the space of interval-constrained partial programs.

These constraints are imposed by the structure of the search DAG, which is a extended version of the search DAG over partial programs. Below, we first describe the children function for interval-constrained partial programs; the concretization function is constructed from the children function.

子节点函数。接下来，我们描述区间约束部分程序的子节点˜$\mathrm{children}(\tilde{p}) \subseteq \mathrm{P}$。直观地说，如果部分程序 ˆ$p$中的洞 N可以用常量值$c \in \mathrm{C}$ 填充（即存在产生式 $N \to c$）以得到 ˆp'，则 ˆ$p' \in \mathrm{children}(\hat{p})$。相比之下，对于区间约束部分程序 $\tilde{p}$，我们包含一个用 $[-\infty, \infty]$ 标注 N 的子节点。然后，

随后的子节点可以进一步拆分区间约束，以获得更细粒度的区间约束 $\hat{}$（具体常量值 c 可以用约束 $(c, c) \in \mathsf{C}$ 表示）。为了形式化这一概念，我们首先分离出常量的产生式。

Definition 4.6. A production $r \in R$ is *constant* if it has the form $r = N \to c$ for some $c \in C$.

Now, we can partition R into the set R_C of constant productions and its complement $\hat{R} = R \setminus R_C$, which we call *non-constant productions*. Next, given an interval-constrained partial program $\tilde{p} = (\hat{p}, \kappa)$, its holes are simply the holes of the underlying partial program $\hat{p}$: $\text{holes}(\tilde{p}) = \text{holes}(\hat{p})$. Then, we partition these holes into *unannotated holes* and *annotated holes*—i.e.,

$$\text{holes}_\varnothing((\hat{p}, \kappa)) = \{i \in \text{holes}(\hat{p}) \mid \kappa(i) = \varnothing\}$$
$$\text{holes}_C((\hat{p}, \kappa)) = \{i \in \text{holes}(\hat{p}) \mid \kappa(i) \neq \varnothing\},$$

respectively. Finally, we define three kinds of children for an interval-constrained partial program $\tilde{p}$. First, we include children obtained by filling an unannotated hole with a non-constant production:

$$\text{children}_\varnothing(\tilde{p}) = \{(\hat{p}', \kappa') \in \tilde{\mathcal{P}} \mid \exists i \in \text{holes}_\varnothing(\tilde{p}), r \in \hat{R} \,.\, \hat{p} = \text{fill}(\hat{p}, i, r) \wedge \kappa' = \text{repair}(\kappa; \hat{p}, i, r)\}.$$

These children are the same as the children constructed in the original search DAG over partial programs. Here, $\text{repair}(\kappa; \hat{p}, i, r)$ "repairs" κ by accounting for how the indices in $\hat{p}$ change after applying production r to fill hole i in $\hat{p}$. In particular, this operation changes the indices of nonterminals in $\hat{p}$; κ' accounts for these changes without modifying the annotations themselves. Formally, if $\hat{p}' = \text{fill}(\hat{p}, N_i \to M_1 \ldots M_h, i)$, with $\hat{p} = N_1 \ldots N_i \ldots N_k$ and $\hat{p}' = N_1 \ldots M_1 \ldots M_h \ldots N_k$, then we have

$$\kappa'(j) = \begin{cases} \kappa(j) & \text{if } j < i \\ \varnothing & \text{if } i \leq j \leq h \\ \kappa(j - h + 1) & \text{if } j > h. \end{cases}$$

In particular, κ' includes the same annotations as κ.

Second, we consider children obtained by filling an unannotated hole with the interval $[-\infty, \infty]$:

$$\text{children}_\infty(\tilde{p}) = \{(\hat{p}', \kappa') \in \tilde{\mathcal{P}} \mid \exists i \in \text{holes}_\varnothing(\tilde{p}) \,.\, \hat{p}' = \hat{p} \wedge \kappa'(i) = [-\infty, \infty]\}.$$

In other words, the partial program $\hat{p}$ remains unchanged, but we introduce an annotation onto one of the previously unannotated holes of $\tilde{p}$.

Third, we consider children obtained by replacing an annotation with a tighter annotation:

$$\text{children}_C(\tilde{p}) = \{(\hat{p}', \kappa') \in \tilde{\mathcal{P}} \mid \exists i \in \text{holes}_C(\tilde{p}) \,.\, \hat{p}' = \hat{p} \wedge \text{subset}(\kappa'(i), \kappa(i))\}.$$

Here, $\text{subset}(\hat{c}, \hat{c}')$ checks whether $\hat{c} = (c_0, c_1)$ is a *strict* subset of $\hat{c}' = (c_0', c_1')$, that is, $c_0' \leq c_0$ and $c_1 \leq c_1'$, and one of these inequalities is strict; equivalently, $[c_0, c_1] \subsetneq [c_0', c_1']$.

Finally, our overall search DAG is defined by the union of these children:

$$\text{children}(\tilde{p}) = \text{children}_\varnothing(\tilde{p}) \cup \text{children}_\infty(\tilde{p}) \cup \text{children}_C(\tilde{p}). \tag{8}$$

Note that by defining children in this way, there may be infinitely many children; in addition, multiple children may cover the same concrete program, leading to redundancy in the search DAG. Practical implementations can restrict to a finite subset of these children as long as they satisfy (4)—i.e., the union of the concrete programs in the children of $\tilde{p}$ cover all concrete programs in $\tilde{p}$. In addition, these children are ideally chosen so the overlap is minimal.

具体化函数。回想一下，具体化函数 $\gamma(\tilde{p})$包含由 $\tilde{p}$表示的具体程序 p。我们采用基于子节点函数来定义 γ 的方法——即，$p \in \gamma(\tilde{p})$，当且仅当存在一个序列 $\tilde{p}_1$、……、$\tilde{p}_n = p$，使得 $\tilde{p}_1 = \tilde{p}$、$\tilde{p}_n = p$，并且对于所有 $1 < j < n$，都有 $\tilde{p}_{j+1} \in \text{children}(\tilde{p}_j)$。

4.5 Interval Transformers for Partial Programs with Interval Constraints

接下来，我们将介绍如何为带区间约束的部分程序实现抽象解释。抽象解释通常针对程序输入进行，而在我们的情形中，它针对的是程序常量。我们假设所有组件 $f \in \mathcal{F}$ 都有一个ˆ ˆ 抽象变换器 f，且目标函数 ϕ_0 有一个抽象变换器 ϕ_0 （如果它们是单调的，则可以按照第4.2节和第4.3节中的方法定义其抽象变换器）。

First, we modify the grammar of programs so that the constants C are replaced by abstract values $(c_0, c_1) \in \hat{C}$. While the concrete semantics cannot be applied to these programs, we will define abstract semantics for them. Now, given a generalized partial program $\tilde{p} = (\hat{p}, \kappa)$, for each unannotated hole $i \in \text{holes}_{\varnothing}(\tilde{p})$, we replace the corresponding nonterminal N_i in $\hat{p}$ with the abstract value $(-\infty, \infty)$, and for each annotated hole $i \in \text{holes}_C(\tilde{p})$, we replace the corresponding nonterminal N_i with the annotation $\kappa(i)$. Finally, we replace any constant c in $\hat{p}$ with the abstract value (c, c). Once we have performed this transformation, we can define the following abstract semantics for $\hat{p}$:

$$[\![X]\!]^{\#}(x) = \alpha(x), \quad [\![\hat{c}]\!]^{\#}(x) = \hat{c}, \quad [\![f(p_1, ..., p_k)]\!]^{\#}(x) = \hat{f}([\![p_1]\!]^{\#}(x), ..., [\![p_k]\!]^{\#}(x)),$$

Now, we can combine $[\![\cdot]\!]^{\#}$ with $\hat{\phi}_0$ to obtain the abstract objective $[\![\cdot]\!]^{\#}_{\phi}$:

$$[\![(\hat{p}, \kappa)]\!]^{\#}_{\phi}(Z) = \hat{\phi}_0\left(\{([\![\hat{p}]\!]^{\#}(x), y)\}_{(x,y)\in Z}\right).$$

In other words, we apply the abstract semantics $[\![\hat{p}]\!]^{\#}$ to each input x, obtain the corresponding abstract output $\hat{y}'$, and apply $\hat{\phi}_0$ to the resulting set $\{(\hat{y}', y)\}$.

5 IMPLEMENTATION

We instantiate our framework for two different domain specific languages (DSLs):

- **NEAR DSL:** The NEAR language for the CRIM dataset (Section 5.1). The motivating example described in Section 2 is derived from this setting.
- **Quivr DSL:** A query language over video trajectories, which uses constructs similar to regular expressions to match trajectories (Section 5.2).

Although both of these DSLs process sequence data, their computation models are quite different: NEAR focuses on folding combinators over the inputs, whereas Quivr's primary operations reduce to matrix multiplication.

For both DSLs, we refine the definition of children compared to Equation 8. In particular, in Section 4.4, when defining the search space over programs with interval constraints on holes, children_C is defined such that the children of $[a, b]$ are *all* of its strict subintervals. As discussed there, this means that each node may have infinitely many children, which is impractical for an implementation. Instead, our implementation splits intervals into just two children—i.e., the children of $[a, b]$ are $[a, (a+b)/2]$ and $[(a+b)/2, b]$. These intervals partition $[a, b]$, so all concrete programs are still contained in the search space, retaining soundness. This splits an interval into child intervals of equal length, but with additional domain-knowledge, other choices could be made (e.g. with a prior distribution over parameters, splitting into intervals of equal probability mass).

Finally, we also describe how we construct the abstract transformer for the F_1 score, which is commonly used as the objective function in binary classification problems (Section 5.3).

5.1 NEAR DSL for Trajectory Labeling

在 NEAR DSL 中 [Shah 等人 2020], 程序的输入是一个特征化轨迹，即由一系列特征向量组成的序列 $x \in (\mathbb{R}^n)^*$，其中 n 是每帧特征向量的维度。输出是一个标签序列 $y \in \{\mathbf{t}, \mathbf{f}\}^*$ 其长度与输入相同，其中 $y[t] = \mathbf{t}$ 表示一帧

表现出给定行为， $y[t] = \mathbf{f}$则表示其他情况（即该任务是在帧级进行二元分类）。

语法。该领域特定语言有三类表达式，分别由 G 中对应的非终结符编码：(i) vv 表示将特征向量映射为实数值的函数（例如 map 的主体），(ii)ℓv 表示将列表映射为实数值的函数（例如 fold），(iii) $\ell\ell$ 表示将列表映射为列表的函数（例如 map）。具体而言，该领域特定语言包含以下产生式：

$$
\begin{aligned}
vv &::= z_i \mid z_f \mid c \in \mathbb{R} \mid vv + vv \mid vv \cdot vv \mid \text{ite}(vv, vv, vv) \qquad (i \in [n]) \\
\ell v &::= \text{fold}(vv) \mid \text{ite}(\ell v, \ell v, \ell v) \\
\ell\ell &::= \text{map}(vv) \mid \text{mapprefix}(\ell v) \mid \text{ite}(\ell v, \ell\ell, \ell\ell),
\end{aligned}
$$

该领域特定语言的语法采用组合子风格，因此省略了 λ。具体而言，由 vv 编码的表达式是专为在 map 或 fold 等高阶函数中使用而设计的组合子。这些组合子应用于输入列表中的各个元素，其中 z_i 是一个变量，表示当前特征向量 $z = x[t]$中的第 i个元素，而z_f是 fold 内部使用的特殊符号，用于表示累积的运行状态。

开始符号是 $\ell\ell$。该领域特定语言的结构使得特征向量列表 x 被映射为等长的实数值列表 r。每个实数值输出 $r[t]$ 隐式编码了标签

$$
y[t] = \begin{cases} \mathbf{t} & \text{if } r[t] \geq 0 \\ \mathbf{f} & \text{otherwise.} \end{cases}
$$

The running example involved programs in a toy DSL of the form $\text{map}(d \leq c)$. In the NEAR DSL, this would be represented as $\text{map}(-1 \cdot z_4 + c)$, as "distance between mice" is the 4th feature, and the label will be obtained by comparing the program output to 0.

Semantics. We let $V = \mathbb{R}^n$ (where n is the dimension of the feature space) be the space of feature vectors, $L = V^*$ is the space of trajectories, and $K = \mathbb{R}^*$ is the space of output sequences. Then, the nonterminal vv in this DSL has semantics $[\![vv]\!] : V \times \mathbb{R} \to \mathbb{R}$, where

$$
\begin{aligned}
[\![z_i]\!](v, s) &\coloneqq v_i \\
[\![z_f]\!](v, s) &\coloneqq s \\
[\![c]\!](v, s) &\coloneqq c \\
[\![vv_1 + vv_2]\!](v, s) &\coloneqq [\![vv_1]\!](v, s) + [\![vv_2]\!](v, s) \\
[\![vv_1 \cdot vv_2]\!](v, s) &\coloneqq [\![vv_1]\!](v, s) \cdot [\![vv_2]\!](v, s) \\
[\![\text{ite}(vv_1, vv_2, vv_3)]\!](v, s) &\coloneqq \text{if } ([\![vv_1]\!](v, s) \geq 0) \text{ then } [\![vv_2]\!](v, s) \text{ else } [\![vv_3]\!](v, s).
\end{aligned}
$$

Next, the nonterminal ℓv in this DSL has semantics $[\![\ell v]\!] : V^* \to \mathbb{R}$, where

$$
\begin{aligned}
[\![\text{fold}(vv)]\!](\ell) &\coloneqq \text{fold}(\lambda v \lambda s.\ [\![vv]\!](v, s), \ell, 0) \\
[\![\text{ite}(\ell v_1, \ell v_2, \ell v_3)]\!](\ell) &\coloneqq \text{if } ([\![\ell v_1]\!](\ell) \geq 0) \text{ then } [\![\ell v_2]\!](\ell) \text{ else } [\![\ell v_3]\!](\ell).
\end{aligned}
$$

Here, $\text{fold} : (V \to \mathbb{R} \to \mathbb{R}) \to V^* \to \mathbb{R} \to \mathbb{R}$ is standard, and passes the intermediate value as the s argument to vv. The nonterminal $\ell\ell$ in this DSL has semantics $[\![\ell\ell]\!] : L \to K$, where

$$
\begin{aligned}
[\![\text{map}(vv)]\!](\ell) &\coloneqq \text{map}(\lambda v.\ [\![vv]\!](v, 0), \ell) \\
[\![\text{mapprefixes}(\ell v)]\!](\ell) &\coloneqq \text{map}(\lambda \ell'.\ [\![\ell v]\!](\ell'), \text{prefixes}(\ell)) \\
[\![\text{ite}(\ell v, \ell\ell_1, \ell\ell_2)]\!](\ell) &\coloneqq \text{if } ([\![\ell v]\!](\ell) \geq 0) \text{ then } [\![\ell\ell_1]\!](\ell) \text{ else } [\![\ell\ell_2]\!](\ell).
\end{aligned}
$$

Here, map is standard[3] and

$$\text{prefixes} : (x_1, \ldots, x_n) \mapsto ((x_1), (x_1, x_2), (x_1, x_2, x_3), \ldots (x_1, \ldots, x_n)).$$

最后，如上所述，通过对 $[\![\ell\ell]\!](\ell)$ 进行阈值处理可得到标签 y。也就是说，我们令$[\![\ell\ell]\!]^b\colon L \to \{\mathbf{t}, \mathbf{f}\}^*$ 表示标签

$$([\![\ell\ell]\!]^b(\ell))[t] = \begin{cases} \mathbf{t} & \text{if } ([\![\ell\ell]\!](\ell))[t] \geq 0 \\ \mathbf{f} & \text{otherwise.} \end{cases}$$

^^ 抽象语义。我们用常见的实数区间 R 来抽象 R。我们用 $(\mathbf{R})^*$（实数区间的乘积）来抽象 K。加法是单调的，并且可以证明乘法 $[a_1, b_1] \cdot [a_2, b_2]$ 可由 $[\min(a_1b_1, a_1b_2, a_2b_1, a_2b_2), \max(a_1b_1, a_1b_2, a_2b_1, a_2b_2)]$ 抽象。我们可以将 $\text{ite}(c, a, b)$ 表示为 $\mathbb{1}[c \geq 0] \cdot a + \mathbb{1}[c < 0] \cdot b$，因此可以通过加法、乘法，以及将 $\mathbb{1}[[\mathbf{f}, \mathbf{f}]] = [0, 0]$, $\mathbb{1}[[\mathbf{f}, \mathbf{t}]] = [0, 1]$ 和 $\mathbb{1}[[\mathbf{t}, \mathbf{t}]] = [1, 1]$ 映射的抽象来抽象它。

搜索空间。 vv 上的搜索空间由于交换律、结合律和分配律而存在冗余，从而不必要地阻碍了搜索。相反，我们使用一个规范化 版本，其中vv 表达式被约束为积之和（完全分配），并且在积之和中忽略交换律和结合律。实质上，我们只考虑多项式，其中变量是特征 z_i、折叠变量 ^zf，以及针对每个 vv 的指示变量 $\mathbb{1}[vv \geq 0]$（以保持 ite 的表达能力）。此外，我们只考虑 $[-1, 1]$ 中的常量，并且还会对数据集进行规范化，使每个特征 z_i 都位于 $[-1, 1]$ 中。

我们为程序定义了大小的概念，以便限制搜索空间：ite、map 和 mapprefix 的大小为 1，fold 的大小为 0（因为 mapprefix 必须包含一个 fold，且其大小已经为 1）。多项式中的每个单项式大小为 1，单项式中的每个多项式变量大小也为 1。在给定的多项式中，指示变量的大小采用摊销计算：每个嵌套的 vv 大小为 1，并生成一个新的多项式变量。

5.2 Quivr DSL For Trajectory Queries

In the Quivr DSL [Mell et al. 2023], the program input is a featurized trajectory, which is a sequence $x \in (\mathbb{R}^n)^*$ as before. However, the output is now a single label $y \in \{\mathbf{t}, \mathbf{f}\}$ for the entire trajectory. This DSL is designed to allow the user to select trajectories that satisfy certain properties—e.g. the user may want to identify all cars that make a right turn at a certain intersection in a traffic video.

Syntax. This DSL is based on the Kleene algebra with tests [Kozen 1997], which, intuitively, are regular expressions where the "characters" are actually predicates. Its syntax is

$$\begin{aligned} Q &::= f \mid g(C) \mid Q\,;Q \mid Q \wedge Q && (f \in \mathcal{F}_\varnothing, g \in \mathcal{F}_C) \\ C &::= c && (c \in \mathbb{R}) \end{aligned}$$

where $\mathcal{F}_\varnothing$ and $\mathcal{F}_C$ are given sets of domain-specific predicates, the latter of which have constants $c \in \mathbb{R}$ that need to be chosen by the synthesizer.

Semantics. Expressions in this DSL denote functions mapping sequences of feature vectors $x \in (\mathbb{R}^n)^*$ to whole-sequence labels ($\{\mathbf{t}, \mathbf{f}\}$), defined as

$$\begin{aligned} [\![f]\!](x) &:= f(x) \\ [\![g(c)]\!](x) &:= g(x) \geq c \\ [\![Q_1 \wedge Q_2]\!](x) &:= [\![Q_1]\!](x) \wedge [\![Q_2]\!](x) \end{aligned}$$

[3]But note that z_f is treated as 0 if it appears inside an expression not within fold

$$[\![Q_1 \,;\, Q_2]\!](x) \;\coloneqq \bigvee_{k=0}^{n} [\![Q_1]\!](x_{0:k}) \wedge [\![Q_2]\!](x_{k:n})$$

Each predicate $f \in \mathcal{F}_\varnothing$ has type $f : (\mathbb{R}^n)^* \to \{\mathbf{t}, \mathbf{f}\}$, and indicates whether x matches f, and each predicate $g \in \mathcal{F}_C$ has type $g : (\mathbb{R}^n)^* \to \mathbb{R}$, and produces a real-valued score thresholded at a given constant c to indicate whether x matches g.

Abstract Semantics. Note that under the standard orders for reals $\mathbb{R}$ and Booleans $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$ (i.e., $\mathbf{f} < \mathbf{t}$), the semantics $[\![g(c)]\!](x)$ is monotone decreasing with respect to c. Furthermore, both conjunction and disjunction are monotone increasing in their inputs. Thus, we can use our interval transformers for monotone functions from Section 4.5. In particular, recall that $\hat{\mathbb{R}}$ is the abstract domain of real intervals, and let $\hat{\mathbb{B}}$ be the abstract domain of Boolean intervals

$$\hat{\mathbb{B}} = \{(\mathbf{f}, \mathbf{f}), (\mathbf{f}, \mathbf{t}), (\mathbf{t}, \mathbf{t})\}.$$

Then, we use the abstract transformers

$$g(x) \geq (c_0, c_1) = \begin{cases} (\mathbf{f}, \mathbf{f}) & \text{if } g(x) < c_0 \\ (\mathbf{f}, \mathbf{t}) & \text{if } c_0 \leq g(x) < c_1 \\ (\mathbf{t}, \mathbf{t}) & \text{if } c_1 \leq g(x) \end{cases}$$

$$\hat{x}_1 \wedge \hat{x}_2 = \begin{cases} (\mathbf{t}, \mathbf{t}) & \text{if } \hat{x}_1 = (\mathbf{t}, \mathbf{t}) \wedge \hat{x}_2 = (\mathbf{t}, \mathbf{t}) \\ (\mathbf{f}, \mathbf{f}) & \text{if } \hat{x}_1 = (\mathbf{f}, \mathbf{f}) \vee \hat{x}_2 = (\mathbf{f}, \mathbf{f}) \\ (\mathbf{f}, \mathbf{t}) & \text{otherwise} \end{cases}$$

$$\hat{x}_1 \vee \hat{x}_2 = \begin{cases} (\mathbf{t}, \mathbf{t}) & \text{if } \hat{x}_1 = (\mathbf{t}, \mathbf{t}) \vee \hat{x}_2 = (\mathbf{t}, \mathbf{t}) \\ (\mathbf{f}, \mathbf{f}) & \text{if } \hat{x}_1 = (\mathbf{f}, \mathbf{f}) \wedge \hat{x}_2 = (\mathbf{f}, \mathbf{f}) \\ (\mathbf{f}, \mathbf{t}) & \text{otherwise.} \end{cases}$$

Since the concrete semantics are defined in terms of these operators, composing their abstract versions gives an abstract semantics.

5.3 Abstract F_1 Score

In Section 4, we described how to construct abstract transformers for monotone functions, which covers most components in these DSLs. However, many objectives commonly used in practice, such as the F_1 score, are non-monotone. For non-monotone objectives, we need to provide a custom abstract transformer that overapproximates their concrete semantics for the interval domain. We now describe how to do so for the F_1 score.

Let W be the multiset of outcomes of the form (y', y) where y' is the prediction and y is the ground truth. Then the F_1 score is given by:

$$TP(W) \coloneqq \sum_{(y',y) \in W} \mathbb{1}[y = \mathbf{t} \wedge y' = \mathbf{t}]$$

$$FP(W) \coloneqq \sum_{(y',y) \in W} \mathbb{1}[y = \mathbf{f} \wedge y' = \mathbf{t}]$$

$$F_1(W) \coloneqq 2 \cdot \frac{TP(W)}{TP(W) + FP(W) + |W^+|},$$

where $|W^+| = |\{(y', y) \in W \mid y = \mathbf{t}\}|$, TP is the number of true positives, and FP is the number of false positives. Note that TP and FP are monotone in y', so we can use the corresponding interval

transformers:

$$TP^{\#}(\hat{W}) \coloneqq \sum_{(\hat{y}',y)\in\hat{W}} \begin{cases} [1,1] & \text{if } y = \mathbf{t} \wedge \hat{y}' = (\mathbf{t},\mathbf{t}) \\ [0,1] & \text{if } y = \mathbf{t} \wedge \hat{y}' = (\mathbf{f},\mathbf{t}) \\ [0,0] & \text{if } y = \mathbf{t} \wedge \hat{y}' = (\mathbf{f},\mathbf{f}) \end{cases}$$

$$FP^{\#}(\hat{W}) \coloneqq \sum_{(\hat{y}',y)\in\hat{W}} \begin{cases} [1,1] & \text{if } y = \mathbf{f} \wedge \hat{y}' = (\mathbf{t},\mathbf{t}) \\ [0,1] & \text{if } y = \mathbf{f} \wedge \hat{y}' = (\mathbf{f},\mathbf{t}) \\ [0,0] & \text{if } y = \mathbf{f} \wedge \hat{y}' = (\mathbf{f},\mathbf{f}). \end{cases}$$

Letting $TP^{\#}(W) = [a_1, b_1]$ and $FP^{\#}(W) = [a_2, b_2]$, and noting that for positive numbers we can abstract division as $[a_1, b_1]/[a_2, b_2] = \left[\frac{a_1}{b_2}, \frac{b_1}{a_2}\right]$, a naïve strategy of applying abstractions for $+$, $\cdot$, and / leads to the abstract F_1 score

$$F_1^{\#}(W) \coloneqq 2 \cdot \frac{[a_1, b_1]}{[a_1, b_1] + [a_2, b_2] + |W^+|} = 2 \cdot \left[\frac{a_1}{b_1 + b_2 + |W^+|}, \frac{b_1}{a_1 + a_2 + |W^+|}\right],$$

where $|W^+|$ is independent of y', so we can treat it as a constant. However, this abstraction is very loose, and we found it not to be useful in practice—e.g. it can be as loose as $[0, 2]$ even though F_1 scores never exceed 1. Instead, we can rewrite

$$F_1(W) = 2 \cdot \frac{\frac{TP(W)}{FP(W)+|W^+|}}{\frac{TP(W)}{FP(W)+|W^+|} + 1} = 2 \cdot L\left(\frac{TP(W)}{FP(W) + |W^+|}\right)$$

where $L(x) = \frac{x}{x+1}$ is monotone, which leads to the abstract F_1 score

$$F_1^{\#}(W) \coloneqq 2 \cdot L\left(\frac{[a_1, b_1]}{[a_2, b_2] + |W^+|}\right) = 2 \cdot L\left(\left[\frac{a_1}{b_2 + |W^+|}, \frac{b_1}{a_2 + |W^+|}\right]\right)$$
$$= 2 \cdot \left[\frac{a_1}{a_1 + b_2 + |W^+|}, \frac{b_1}{b_1 + a_2 + |W^+|}\right].$$

6 EXPERIMENTS

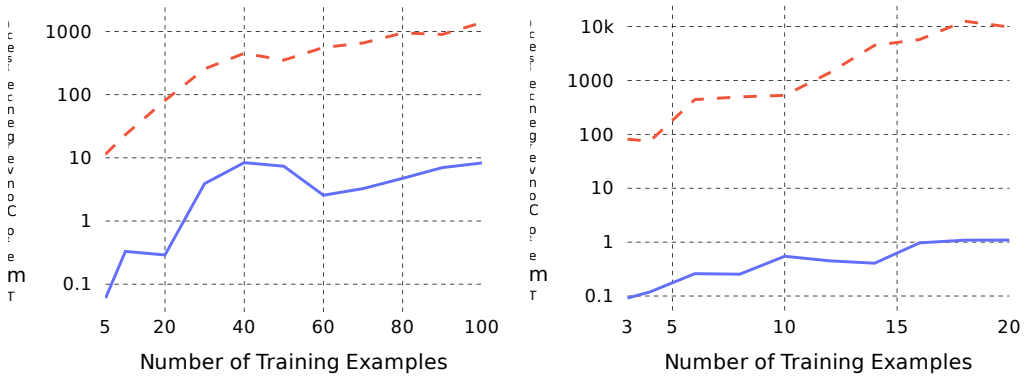
We experimentally evaluate our approach in the context of the NEAR and Quivr DSLs described in Section 5. We demonstrate that our synthesizer outperforms two synthesis baselines in scalability of running time: Metasketches [Bornholt et al. 2016], an optimal synthesizer based on SMT solvers (Section 6.2), as well as an ablation that uses breadth-first search instead of our abstract interpretation based heuristic and A^* search (Section 6.3).

6.1 Experimental Setup

Benchmarks. We consider two different neurosymbolic program synthesis benchmarks, based on the DSLs described in Section 5:

- **CRIM13:** The NEAR [Shah et al. 2020] DSL applied to two tasks in the CRIM13 dataset [Burgos-Artizzu et al. 2012] "sniff" (A) and "other" (B). This dataset consists of featurized videos of two mice interacting in an enclosure, with 12,404 training examples with 100 frames each.
- **Quivr:** The Quivr [Mell et al. 2023] DSL, applied to the 17 tasks that they evaluate on. Of these, 6 tasks are on the MABe22 [Sun et al. 2022] dataset, a dataset of interactions between 3 mice, and 10 tasks are on the YTStreams [Bastani et al. 2020] dataset, a dataset of video from fixed-position traffic cameras.

计算。 我们在一台英特尔至强 Gold 6342 CPU（2.80GHz，36 核/72 线程）上运行了所有实验。我们的实现使用 PyTorch，这是一个提供快速矩阵运算的库。



(a) Quivr, task G (b) CRIM13, task A

图 2. 对于两种不同的任务，随着训练数据集大小的变化，我们的方法（蓝色实线）和 SMT 求解器（红色虚线）识别最优程序并证明其最优性所需的时间（秒，对数刻度）。

6.2 与 Metasketches 的比较

Metasketches 使用 SMT 求解器进行最优综合：除了正确性规范外，还会添加一个 SMT 约束，要求程序得分高于迄今为止观察到的最佳得分。因此，如果 SMT 求解器返回“SAT”，就说明找到了一个更优的程序，随后便会重复这一过程。请注意，在我们的设定中，不存在正确性规范，因此，获得更高得分是唯一的 SMT 约束。

一种类似但我们发现更有效的策略，是对目标得分进行二分搜索：假设目标函数在 $[0, 1]$, 中，我们询问 SMT 求解器是否存在得分至少为 $1/2$ 的程序；如果返回“SAT”，我们就询问是否存在得分至少为 $3/4$ 的程序；如果返回“UNSAT”，则询问是否存在得分至少为 $1/4$ 的程序，依此类推。我们的实现使用 Z3 [De Moura 和 Bjørner 2008] SMT 求解器。为进行更公平的比较，在本实验中我们将 PyTorch 限制为仅使用一个 CPU 核心。

两种方法都运行到收敛至精确的最优程序为止。图 2 显示，随着训练数据集中轨迹数量的增加，SMT 求解器的扩展性非常差。虽然在处理三到四条轨迹时具有竞争力，但我们希望在包含数百或数千条轨迹的数据集上进行评估。

6.3 Comparison to Breadth-First Search

Next, to show the benefit of the search heuristic, we compare against an ablation which ignores the heuristic and does breadth-first search.

At any point in the search process, there is a heap of search nodes, each of which has a lower and upper bound of F_1 scores reachable from it. Rather than using the lower bound from the abstract objective value, we instead evaluate the concrete program whose parameters are the midpoint of the hyper-rectangle of abstract parameters, to get a concrete objective value; this is a better lower-bound, and it is cheap to compute. The greatest of these lower bounds provides a lower bound on the optimal F_1 score, and the greatest of these upper bounds provides an upper bound on the optimal F_1 score. Note that if the lower and upper bounds are equal, they equal the true optimal F_1 score, and search terminates.

To make search tractable, on the CRIM13 benchmarks we consider only expressions with structural cost at most 4. Note that this rules out “ite” expressions, but performs well in practice. On the

Table 1. Best F_1 score found and range between upper and lower bounds, at a particular time during search, for different tasks and different algorithms. "CA" and "CB" are the NEAR CRIM13 queries, and "QA" through "QQ" are the Quivr queries. "H" is A^* search and "B" is breadth-first search. For each task and each time, the best F_1 score is bolded and the smallest range is bolded. A range of 0 implies that search has converged to the optimal F_1 score.

Setting		**10 sec**		**30 sec**		**1 min**		**2 min**		**5 min**		**10 min**	
CA	H	0.21	(0.79)	0.21	(0.79)	0.21	(0.79)	**0.52**	**(0.14)**	**0.53**	**(0.03)**	**0.53**	**(0.00)**
	B	0.21	(0.79)	0.21	(0.79)	0.21	(0.79)	0.22	(0.78)	0.46	(0.54)	0.50	(0.50)
CB	H	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	**0.80**	**(0.05)**	**0.80**	**(0.04)**	**0.80**	**(0.01)**
	B	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)
QA	H	0.12	(0.88)	**0.77**	**(0.23)**	**0.77**	**(0.23)**	**0.77**	**(0.23)**	**0.77**	**(0.23)**	**0.77**	**(0.23)**
	B	0.12	(0.88)	0.26	(0.74)	0.26	(0.74)	0.26	(0.74)	0.26	(0.74)	0.28	(0.72)
QB	H	0.27	(0.73)	0.31	(0.69)	**0.52**	**(0.48)**	**0.52**	**(0.48)**	**0.52**	**(0.48)**	**0.52**	**(0.48)**
	B	0.27	(0.73)	**0.40**	**(0.60)**	0.40	(0.60)	0.40	(0.60)	0.42	(0.58)	0.50	(0.50)
QC	H	0.06	(0.94)	0.14	(0.86)	**0.30**	**(0.70)**	**0.33**	**(0.67)**	**0.38**	**(0.62)**	**0.38**	**(0.62)**
	B	0.06	(0.94)	**0.25**	**(0.75)**	0.25	(0.75)	0.26	(0.74)	0.26	(0.74)	0.26	(0.74)
QD	H	0.04	(0.96)	**0.25**	**(0.75)**	**0.25**	**(0.75)**	**0.25**	**(0.75)**	**0.40**	**(0.60)**	**0.44**	**(0.56)**
	B	0.04	(0.96)	0.06	(0.94)	0.06	(0.94)	0.06	(0.94)	0.09	(0.91)	0.19	(0.81)
QE	H	0.04	(0.96)	**0.20**	**(0.80)**	**0.44**	**(0.56)**	**0.44**	**(0.56)**	**0.44**	**(0.56)**	**0.44**	**(0.56)**
	B	0.04	(0.96)	0.06	(0.94)	0.06	(0.94)	0.10	(0.90)	0.10	(0.90)	0.15	(0.85)
QF	H	0.38	(0.62)	**0.78**	**(0.22)**	**0.78**	**(0.22)**	**0.78**	**(0.22)**	**0.78**	**(0.22)**	**0.78**	**(0.22)**
	B	0.38	(0.62)	0.42	(0.58)	0.42	(0.58)	0.55	(0.45)	0.56	(0.44)	0.60	(0.40)
QG	H	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)
	B	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)
QH	H	0.74	(0.26)	0.74	(0.26)	0.74	(0.26)	0.74	(0.26)	0.74	(0.26)	0.74	(0.26)
	B	**0.78**	**(0.22)**	**0.78**	**(0.22)**	**0.78**	**(0.22)**	**0.78**	**(0.22)**	**0.78**	**(0.22)**	**0.78**	**(0.22)**
QI	H	0.75	(0.25)	**0.80**	**(0.20)**	**0.80**	**(0.20)**	**0.80**	**(0.20)**	**0.80**	**(0.20)**	**0.80**	**(0.20)**
	B	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)
QJ	H	0.64	(0.36)	0.64	(0.36)	0.64	(0.36)	0.64	(0.36)	0.64	(0.36)	0.64	(0.36)
	B	0.64	(0.36)	**0.73**	**(0.27)**	**0.73**	**(0.27)**	**0.73**	**(0.27)**	**0.73**	**(0.27)**	**0.73**	**(0.27)**
QK	H	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)
	B	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)	0.75	(0.25)
QL	H	0.71	(0.29)	0.91	(0.09)	0.91	(0.09)	0.91	(0.09)	0.91	(0.09)	0.91	(0.09)
	B	0.71	(0.29)	0.91	(0.09)	0.91	(0.09)	0.91	(0.09)	0.91	(0.09)	0.91	(0.09)
QM	H	0.73	(0.27)	**0.92**	**(0.08)**	**0.92**	**(0.08)**	**0.92**	**(0.08)**	0.92	(0.08)	0.92	(0.08)
	B	0.73	(0.27)	0.80	(0.20)	0.80	(0.20)	0.80	(0.20)	0.92	(0.08)	0.92	(0.08)
QN	H	0.73	(0.27)	0.73	(0.27)	**0.89**	**(0.11)**	**0.89**	**(0.11)**	**0.89**	**(0.11)**	**0.89**	**(0.11)**
	B	0.73	(0.27)	0.73	(0.27)	0.80	(0.20)	0.80	(0.20)	0.80	(0.20)	0.80	(0.20)
QO	H	0.67	(0.33)	0.67	(0.33)	**0.86**	**(0.14)**	**0.86**	**(0.14)**	**0.86**	**(0.14)**	**0.86**	**(0.14)**
	B	0.67	(0.33)	0.67	(0.33)	0.80	(0.20)	0.80	(0.20)	0.80	(0.20)	0.80	(0.20)
QP	H	0.50	(0.50)	0.50	(0.50)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)
	B	0.50	(0.50)	0.50	(0.50)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)
QQ	H	0.80	(0.20)	0.80	(0.20)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)
	B	0.80	(0.20)	0.80	(0.20)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)	1.00	(0.00)

Quivr benchmarks, we bound the search space in the same way that their paper does, limiting to programs with at most 3 predicates, at most 2 of which have parameters.

We use 100 trajectories from each dataset. For CRIM13, these are randomly sampled from the training set. For Quivr, to ensure that we have some positive examples, because positives are very

sparse on some tasks, we use 2 positive and 10 negative trajectories specially designated in the dataset, and the remaining 88 are sampled randomly from the training set.

表1 展示了搜索过程中不同时间点的当前最佳 F_1 得分（区间下界），以及最优 F_1 得分区间的宽度。在大多数任务上，我们的方法（H）比消融版本（B）更快地达到更高的 F_1 得分，同时区间也更窄。

7 相关工作

神经符号综合。近年来，人们对神经符号综合产生了浓厚兴趣 [Chaudhuri 等 2021], , 其中包括函数式程序的合成 [Gaunt 等，2016；Shah 等，2020；Valkov 等 2018], 、强化学习策略的合成 [Anderson 等，2020；Inala 等 2020],、从非结构化文本中提取数据的程序的合成 [Chen 等，2023，2021；Ye 等 2021], , 以及从视频轨迹中提取数据的程序的合成 [Bastani 等，2021；Mell 等，2023；Shah 等2020]。其中一些方法提出了基于单调性的剪枝策略 [Chen 等，2021；Mell 等 2023], , 但仅适用于特定的领域特定语言。NEAR 是一个基于神经启发式的通用神经符号综合框架 [Shah 等 2020]；然而，该方法无法保证合成出最优程序。据我们所知，我们的工作提出了首个用于神经符号程序最优综合的通用框架。

最优合成。更广泛地说，近年来人们对最优合成产生了兴趣 [Bornholt 等，2016；Smith 和 Albarghouthi 2016],通常侧重于优化程序的性能属性（例如运行时间），而非准确率；超优化是研究尤为广泛的一种应用 [Bansal 和 Aiken，2008年；Massalin，1987；Mukherjee 等，2020；Phothilimthana 等，2016；Sasnauskas 等 2017]。我们的实验表明，我们的方法优于 Bornholt 等[2016], 一种基于 SMT 求解器的通用最优合成框架。此外，也有研究致力于合成能够最大化某个目标函数（以神经网络评分函数表示）的程序 [Ye 等 2021], 但他们没有考虑实值常量；定量目标是句法层面的，而非语义层面的；抽象解释仅用于依据布尔正确性规范进行剪枝，而不用于定量目标。最优合成也被用于合成保障内存安全的最小保护条件 [Dillig 等 2014], 化学反应网络 [Cardelli 等 2017], 以及面向量子计算的最优布局 [Tan 和 Cong 2020]。

Abstract Interpretation for Synthesis. There has been work on leveraging abstract interpretation for pruning portions of the search space in program synthesis [Guria et al. 2023; So and Oh 2017], as well as using abstraction refinement [Wang et al. 2017]; however, these approaches target traditional synthesis. Rather than evaluating an abstract semantics on partial programs, Wang et al. [2017] constructs a data structure compactly representing concrete programs whose abstract semantics are compatible with the input-output examples. However, it is not obvious how their data structure (which targets Boolean specifications) can be adapted to our quantitative synthesis setting.

用于规划的抽象解释。 一条研究路线由 FF 规划器开创 [Hoffmann 和 Nebel 2001],使用抽象语义进行可达性分析以剪枝无效计划 [Gregory 等人 2012年; Hoffmann 2003年; Zhi-Xuan 等人 2022]。然而，除了剪枝无效计划之外，可达性分析并未用于搜索启发式的计算。相反，使用的是诸如 $h_{\max}$ （在从抽象动作的后置条件中去掉删除列表的“松弛模型”中计算最短计划，并输出该计划长度）之类的传统启发式。相比之下，我们使用针对目标函数的抽象变换器提供下界，并直接将其用作搜索启发式。

第二条研究路线 [Marthi 等人 2008年; Vega-Brown 和 Roy 2018] 考虑通过下近似成本函数来计算最优计划。他们假设某一计划的总成本等于该计划中各个动作的成本之和，然后在给定每个动作成本的下界后，将这些下界简单相加，从而得到整体计划成本的下界。该策略对整体成本函数的结构做出了很强的假设，而我们基于抽象解释的方法则无需此类假设。

Another key difference is that we are abstracting over real-valued parameters of partial programs, whereas the above approaches are abstracting over continuous states. Thus, our framework requires a way to iteratively refine the program space (specified by the "children" function), which is absent from their frameworks.

8 CONCLUSION

We have proposed a general framework for synthesizing programs with real-valued inputs and outputs, using A^* search in conjunction with a search heuristic based on abstract interpretation. Our framework searches over a space of generalized partial programs, which represent sets of concrete programs, and uses the search heuristic to establish upper bounds on the objective value of a given generalized partial program. In addition, we propose a natural strategy for constructing abstract transformers for components with monotone semantics. If our algorithm returns a program, then this program is guaranteed to be optimal. Our experimental evaluation demonstrates that our approach is more scalable than existing optimal synthesis techniques. Directions for future work include improving the scalability of our approach and applying it to additional synthesis tasks.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful feedback. This work was supported in part by NSF Award CCF-1910769, NSF Award CCF-1917852, ARO Award W911NF-20-1-0080, and Amazon/ASSET Gift for Research in Trustworthy AI.

DATA AVAILABILITY STATEMENT

An artifact is available [Mell 2023] with our implementation, which reproduces our experimental results (Figure 2 and Table 1) and may be useful for performing synthesis on other trajectory datasets or implementing our algorithm for other DSLs.

REFERENCES

Greg Anderson、Abhinav Verma、Isil Dillig 和 Swarat Chaudhuri。2020年。通过形式化验证的探索进行神经符号强化学习。神经信息处理系统进展33（2020年），6172–6183。Sorav Bansal 和 Alex Aiken。2008年。使用窥孔超优化器进行二进制翻译。。载于 操作系统设计与实现研讨会，第8卷。177–192。Favyen Bastani、Songtao He、Arjun Balasingam、Karthik Gopalakrishnan、Mohammad Alizadeh、Hari Balakrishnan、Michael Cafarella、Tim Kraska 和 Sam Madden。2020年。MIRIS：用于视频的快速目标跟踪查询。载于 *2020*年 *ACM SIGMOD*数据管理国际会议论文集。1907–1921。https://doi.org/10.1145/3318464.3389692 Favyen Bastani、Songtao He、Ziwen Jiang、Osbert Bastani 和 Sam Madden。2021年。SkyQuery：空中无人机视频感知平台。载于 *2021*年 *ACM SIGPLAN*编程语言与软件领域新思想、新范式及反思国际研讨会论文集。56–67。https://doi.org/10.1145/3486607.3486750 James Bornholt、Emina Torlak、Dan Grossman 和 Luis Ceze。2016年。使用元草图优化合成。载于 第*43*届 *ACM SIGPLAN-SIGACT*编程语言原理研讨会论文集。775–788。https: //doi.org/10.1145/2837614.2837666 Xavier P. Burgos-Artizzu、Piotr Dollár、Dayu Lin、David J. Anderson 和 Pietro Perona。2012年。连续视频中的社会行为识别。载于 *2012*年 *IEEE*计算机视觉与模式识别会议。1322–1329。https://doi.org/10. 1109/CVPR.2012.6247817Luca Cardelli、Milan Češka、Martin Fränzle、Marta Kwiatkowska、Luca Laurenti、Nicola Paoletti 和 Max Whitby。2017年。面向化学反应网络的语法引导最优合成。载于 计算机辅助验证：第*29*届国际会议

会议，CAV 2017，德国海德堡，2017年7月24日至28日，论文集，第二部分，第30卷。施普林格，375–395。https://doi.org/10. 1007/978-3-319-63390-9 20_Swarat Chaudhuri、Kevin Ellis、Oleksandr Polozov、Rishabh Singh、Armando Solar-Lezama、Yisong Yue 等。2021年。神经符号编程。《编程语言基础与趋势》7，3（2021年），158–243。https: //doi.org/10.1561/2500000049 Qiaochu Chen、Arko Banerjee、Çağatay Demiralp、Greg Durrett 和 Isil Dillig。2023年。通过语义正则表达式综合进行数据提取。载于OOPSLA。https://doi.org/10.1145/3622863 Qiaochu Chen、Aaron Lamoreaux、Xinyu Wang、Greg Durrett、Osbert Bastani 和 Isil Dillig。2021年。利用神经符号程序合成进行网页问答。载于第42届 ACM SIGPLAN编程语言设计与实现国际会议论文集。328–343。https://doi.org/10.1145/3453483.3454047 Patrick Cousot 和 Radhia Cousot。1977年。抽象解释：通过构造或近似不动点，为程序静态分析建立统一的格模型。载于第4届 ACM SIGACT-SIGPLAN编程语言原理研讨会论文集。238–252。https://doi.org/10.1145/512950.512973 Leonardo De Moura 和 Nikolaj Bjørner。2008年。Z3：一种高效的 SMT 求解器。载于《软件理论与实践》，第14届系统构造与分析工具和算法国际会议（匈牙利布达佩斯）（TACAS'08/ETAPS'08）。施普林格出版社，柏林，海德堡，337–340。https://doi.org/10.1007/978-3-540-78800- 3 24_Thomas Dillig、Isil Dillig 和 Swarat Chaudhuri。2014年。面向内存安全的最优保护条件综合。载于计算机辅助验证：第26届国际会议，CAV 2014，作为维也纳逻辑夏季会议 VSL 2014 的一部分在奥地利维也纳举行，2014年7月18日至22日。论文集，第26卷。施普林格，491–507。https://doi.org/10.1007/978-3-319-08867-9_32Alexander L Gaunt、Marc Brockschmidt、Rishabh Singh、Nate Kushman、Pushmeet Kohli、Jonathan Taylor 和 Daniel Tarlow。2016年。Terpret：一种用于程序归纳的概率编程语言。arXiv预印本 arXiv:1608.04428（2016年）。Peter Gregory、Derek Long、Maria Fox 和 J. Christopher Beck。2012年。模理论规划：扩展规划范式。《自动规划与调度国际会议论文集》22，1（2012年5月），65–73。https://doi.org/10.1609/icaps.v22i1.13505 Sumit Gulwani、Susmit Jha、Ashish Tiwari 和 Ramarathnam Venkatesan。2011 年。无循环程序综合。载于 PLDI'11，2011 年 6 月 4 日至 8 日，美国加利福尼亚州圣何塞（pldi'11，2011年6月4–8日，美国加利福尼亚州圣何塞版）。https://doi.org/10. 1145/1993498.1993506 Sankha Narayan Guria、Jeffrey S Foster 和 David Van Horn。2023年。Absynthe：抽象解释引导的综合。《ACM编程语言学报》7，PLDI（2023年），1584–1607。https://doi.org/10.1145/3591285 Jörg Hoffmann。2003年。Metric-FF规划系统：将"忽略删除列表"转换为数值状态变量。《人工智能研究期刊》20（2003年），291–341。https://doi.org/10.1613/jair.1144 Jörg Hoffmann 和 Bernhard Nebel。2001年。FF规划系统：通过启发式搜索快速生成计划。《人工智能研究期刊》14，1（2001年5月），253–302。https://doi.org/10.1613/jair.855 Jeevana Priya Inala、Yichen Yang、James Paulos、Yewen Pu、Osbert Bastani、Vijay Kumar、Martin Rinard 和 Armando Solar-Lezama。2020年。用于多智能体通信的神经符号变换器。神经信息处理系统进展 33（2020年），13597–13608。Michael J Kearns 和 Umesh Vazirani。1994年。计算学习理论导论。麻省理工学院出版社。https://doi.org/10. 7551/mitpress/3897.001.0001 Dexter Kozen。1997年。带测试的克莱尼代数。《ACM编程语言与系统汇刊》（TOPLAS）19，3（1997年），427–443。https://doi.org/10.1145/256167.256195 Bhaskara Marthi、Stuart Russell 和 Jason Andrew Wolfe。2008年。天使式分层规划：最优在线算法。。载于 ICAPS。Citeseer，222–231。Henry Massalin。1987年。超级优化器：探究最小程序。《ACM SIGARCH计算机体系结构新闻》15，5（1987年），122–126。https://doi.org/10.1145/36177.36194 Stephen Mell。2023年。通过抽象解释实现最优程序综合的工件。https://doi.org/10.5281/zenodo. 10146270 Stephen Mell、Favyen Bastani、Steve Zdancewic 和 Osbert Bastani。2023年。从示例综合轨迹查询。载于《计算机辅助验证》，Constantin Enea 和 Akash Lal（编）。施普林格·自然瑞士，查姆，459–484。https://doi.org/10.1007/978-3-031-37706-8_23 Manasij Mukherjee、Pranav Kant、Zhengyang Liu 和 John Regehr。2020年。基于数据流的剪枝以加速超优化。《ACM编程语言学报》4，OOPSLA，文章177（2020年11月），24页。https://doi.org/10.1145/3428245 Phitchaya Mangpo Phothilimthana、Aditya Thakur、Rastislav Bodik 和 Dinakar Dhurjati。2016年。扩展超优化。载于第二十一届编程语言与操作系统体系结构支持国际会议论文集。297–310。https://doi.org/10.1145/2872362.2872387

Raimondas Sasnauskas、Yang Chen、Peter Collingbourne、Jeroen Ketema、Gratian Lup、Jubi Taneja和John Regehr。2017年。Souper：一种合成式超级优化器。arXiv预印本 arXiv:1711.04422（2017年）。https://doi.org/10.48550/arXiv.1711.04422 Ameesh Shah、Eric Zhan、Jennifer Sun、Abhinav Verma、Yisong Yue和Swarat Chaudhuri。2020年。利用可容许神经启发式学习可微程序。《神经信息处理系统进展》33（2020年），4940–4952。Calvin Smith和Aws Albarghouthi。2016年。MapReduce程序合成。ACM SIGPLAN 通讯 51, 6（2016年），326–340。https://doi.org/10.1145/2980983.2908102 Sunbeom So和Hakjoo Oh。2017年。在静态分析引导下从示例合成命令式程序。载于《静态分析》，Francesco Ranzato（编）。施普林格国际出版，查姆，364–381。https://doi.org/10.1007/978-3-319- 66706-5 18_Armando Solar-Lezama、Liviu Tancau、Rastislav Bodik、Sanjit Seshia和Vijay Saraswat。2006年。有限程序的组合式草图。载于第12届编程语言与操作系统体系结构支持国际会议论文集（美国加利福尼亚州圣何塞）（ASPLOS XII）。美国计算机协会，美国纽约州纽约，404–415。https://doi.org/10.1145/1168857.1168907 Jennifer J. Sun、Brian Geuther、Vivek Kumar、Alice Robie、Catherine Schretter、Keith Sheppard、Dipam Chakraborty、Kristin Branson和Ann Kennedy。2022年。面向多智能体行为表征学习的MABe22基准测试。https://doi.org/10.22002/D1.20186 Jennifer J Sun、Tomomi Karigo、Dipam Chakraborty、Sharada P Mohanty、Benjamin Wild、Quan Sun、Chen Chen、David J Anderson、Pietro Perona、Yisong Yue和Ann Kennedy。2021年。多智能体行为数据集：小鼠双体社会互动。arXiv预印本 arXiv:2104.02710（2021年）。https://doi.org/10.48550/arXiv.2104.02710 Bochen Tan和Jason Cong。2020年。面向量子计算的最优布局合成。载于第39届计算机辅助设计国际会议论文集。1–9。https://doi.org/10.1145/3400302.3415620 Lazar Valkov、Dipak Chaudhari、Akash Srivastava、Charles Sutton和Swarat Chaudhuri。2018年。Houdini：作为程序合成的终身学习。《神经信息处理系统进展》31（2018年）。William Vega-Brown和Nicholas Roy。2018年。面向近最优任务与运动规划的可容许抽象。载于第27届国际人工智能联合会议论文集（瑞典斯德哥尔摩）（IJCAI'18）。AAAI出版社，4852–4859。https://doi.org/10.24963/ijcai.2018/674 Xinyu Wang、Isil Dillig和Rishabh Singh。2017年。使用抽象精化的程序合成。ACM编程语言学报 2, POPL, 第63篇（2017年12月），30页。https://doi.org/10.1145/3158151 Xi Ye、Qiaochu Chen、Isil Dillig和Greg Durrett。2021年。基于多模态规约的最优神经程序合成。载于《计算语言学协会论文集：EMNLP 2021》。1691–1704。https://doi.org/10.18653/v1/2021. findings-emnlp.146 Tan Zhi-Xuan、Joshua B. Tenenbaum和Vikash K. Mansinghka。2022年。基于模型的规划中广义启发式搜索的抽象解释。https://doi.org/10.48550/arXiv.2208.02938 arXiv:2208.02938 [cs.AI]

Received 2023-07-11; accepted 2023-11-07