arXiv:2304.04642v1 [cs.PL] 10 Apr 2023

切蛋糕：一种用于公平分配的语言

NOAH BERTRAM, 康奈尔大学，美国
ALEX LEVINSON,康奈尔大学，美国JUSTIN
HSU,康奈尔大学，美国

经济学中的公平分配 文献探讨如何在多个参与者之间分配资源，使分配达到无嫉妒：每位参与者都能获得自己最喜欢的份额。研究人员针对最常见的情形——参与者想要平分单个物品——开发了各种公平分割协议；然而，这些协议非常复杂，证明其无嫉妒性还需要进行繁琐的分类讨论。

我们提出了 Slice，一种用于公平分配的领域特定语言。该语言中的程序可以转换为编码无嫉妒性及其他目标属性的逻辑公式。随后，这些约束可以交由自动求解器处理。我们证明了约束生成过程是可靠且完备的。我们还介绍了 Slice 的原型实现，并用它自动检查了公平分配文献中若干协议的无嫉妒性。

附加关键词和短语：公平分配、自动验证

1 引言

假设我们想在一组参与者之间分配一件有价值的物品。我们如何确保这种分配是公平的？理想情况下，我们会将物品分成大小相等的份额，并给每位参与者一份。然而在现实中，参与者往往有不同的偏好，对于一件物品是否已经被分成大小相等的份额，他们可能会有不同看法。怎样才算是公平地分配一件物品？

在经济学中，公平分配 领域研究的正是这类问题。研究人员提出了不同的公平性定义，并考察了在各种环境下实现公平分配是否可能——物品可能是可分割的，也可能是不可分割的；根据物品的形状，可能存在几何约束，或者对允许采用的平分方式有所限制。

我们关注研究最为广泛的一种模型，称为公平切蛋糕。在这一模型中，只有一件物品——一块“蛋糕”——它是无限可分的：可以将其切成任意小的份额。每位参与者都有一个估值 函数，为蛋糕的各个子集赋予数值。目标是将蛋糕分成互不相交的份额，并为每位参与者分配一份，使得每位参与者都不会更偏好其他参与者的份额而非自己的份额。这样的分配称为无嫉妒。

1.1 公平切蛋糕协议：挑战与复杂性

虽然在假设参与者估值满足温和条件的情况下，证明公平分配的存在性通常并不困难，但计算公平分配则更具挑战性。公平分配算法首先会根据参与者对当前份额的估值，规定参与者应进行的一系列切割。然后，协议会说明如何将切割所得的份额分配给参与者；例如，参与者可以按照某种固定顺序选择自己最喜欢的份额，或者要求一名代理人取得某个特定份额。

举个简单的例子，假设我们想在两个孩子之间分配一份蛋糕，使每个孩子都不嫉妒另一个孩子的份额。首先，我们可以让其中一个孩子将这份蛋糕平分成他们认为相等的两份。然后，我们可以让另一个孩子挑选自己更喜欢的份额。这一简单协议，也称为 切分—选择 或分割—选择，会产生一种无嫉妒的

作者地址：Noah Bertram，美国康奈尔大学；Alex Levinson，美国康奈尔大学；Justin Hsu，美国康奈尔大学。

© 2023 版权所有归作者所有。本内容为作者版本的作品。此处发布仅供您个人使用。不得再分发。最终版本发表于 *ACM* 编程语言论文集，https://doi.org/10.1145/3591293。

分配：只要两个孩子都遵循协议，他们就不会希望自己得到的是另一个孩子的那份。

虽然这个双代理人协议容易描述和论证，但参与者更多时，情况就困难得多。直到1960 [Robertson 和 Webb 1998], 才提出了针对三名参与者的无嫉妒协议；而针对四名参与者的有界无嫉妒协议，直到 2016 年才作为一个开放问题得到解决，解决者是 Aziz 和 Mackenzie [2016b]；他们的程序最多可进行 203 次切分。随后不久，又提出了适用于任意数量参与者的协议 [Aziz 和 Mackenzie 2016a], ，但它极其复杂——切分次数虽然有限，却受一个五层指数塔所界定，其中 n，这里 n 表示参与者人数。

尽管此类无嫉妒协议是令人印象深刻的成就，但它们并不容易实施。这些协议通常以冗长的伪代码形式规定，其中还经常夹杂着非正式的英语。证明其关键的无嫉妒性属性是一项非常繁琐的任务，需要考虑大量情况。对于大多数协议而言，不可能在证明中详细列出所有细节。除了违反无嫉妒性之外，协议还可能以其他方式出错。例如，协议可能要求某位代理人切过份额的末端，或者将蛋糕的同一部分分配多次。

1.2 我们的工作：自动验证切蛋糕协议

我们探讨如何对公平分割协议的无嫉妒性进行形式化验证。我们的假设是，由于这类性质的证明通常需要繁琐但相当直接的分类讨论，因此自动求解器或许很适合用来处理这类问题。

具体来说，我们采用基于语言的方法。首先，我们为切蛋糕协议开发一种核心语言。我们的语言能够描述 Robertson-Webb 查询模型中的所有协议 [Robertson 和 Webb 1998], ——这是公平分配领域中一种标准的计算模型。我们的操作语义允许不可数的非确定性——这在代理人可能认为多个份额同样理想的情形下是必需的——同时也能对协议错误进行建模，例如切割越界。用我们的语言表达协议，可以消除其伪代码描述中的任何歧义。

其次，我们为这种语言开发了一个约束生成过程。概括来说，我们将程序的操作行为转换为一阶逻辑公式，并以未知的估值函数作为参数。我们证明了约束转换是可靠且完备的，并展示了如何利用这些约束来编码无嫉妒性等性质。

最后，我们在一个名为 Slice 的原型工具中实现了我们的语言。用户可以用我们的语言编写切蛋糕协议，该语言提供用于公平分配的原语操作。我们的工具将程序转换为保证无嫉妒性的约束，然后将其提交给 SMT 求解器。我们的实现还包含一个求值器，可以直接执行已验证的协议。我们通过实现多个切蛋糕协议并自动验证其无嫉妒性来评估该工具。据我们所知，我们的方法是首个对这些协议及其性质进行形式化验证的方法。

概要。 在介绍公平切蛋糕的预备知识并通过一个简单示例说明我们的方法（第 2节）之后，我们将介绍主要的技术贡献：

- 我们开发了一种用于公平切蛋糕协议的核心语言，该语言基于 Robertson-Webb 查询模型（第3节）。
- 我们设计了一种从核心语言到一阶逻辑公式的约束转换。我们证明了这种转换是可靠且完备的，并展示了如何使用我们的约束来编码无嫉妒性及其他目标性质（第4节）。

- 我们在本语言中实现了蛋糕切分文献中提出的几种协议（第 5节）。

- 我们开发了一种原型实现，用于编写、验证和执行协议。我们的工具可以自动验证示例协议是否满足无嫉妒性（第 6 节）。

我们在第7节回顾相关工作，并在第 8 节讨论未来方向。

2 验证公平分配：预备知识与一个简单示例

公平分配是经济学中一个内容丰富且经过深入研究的领域 [Robertson 和 Webb 1998]。我们聚焦于最常见的情形： 公平切蛋糕。

2.1 数学预备知识

蛋糕和份额。 我们假设参与者有兴趣对一个称为蛋糕的无限可分割物品进行分割，该物品被建模为闭单位区间 $[0,1]$。每位参与者获得蛋糕的一个子集，其中允许的子集的集合 P 由不相交区间的有限并组成：

$$P \triangleq \{\langle a_1, b_1\rangle \uplus \cdots \uplus \langle a_k, b_k\rangle \mid 0 \le a_i \le b_i \le 1, k \in \mathbb{N}\},$$

每个区间在任一端点处都可以是开区间或闭区间，即，a, b代表四个集合中的一个：$[a,b],(a,b)$ $[a,b),(a,b]$。我们将 P中的一个元素称为一份额；注意，它可能由多个不相交的区间组成。

参与者和估值。 公平分割协议针对固定数量的有限参与者而设计，其集合记为 A。每个参与者 $a \in \mathsf{A}$ 都有一个 价值函数 $V_a\colon P \to \mathbb{R}$；参与者更偏好估值更高的份额。公平分配文献考虑了关于价值函数的各种标准假设。具体而言，我们将假设以下四个条件：

归一化。 整个蛋糕的值为 1： $V([0,1]) = 1$。
非负性。 任意份额 p的值均为非负数： $V(p) \ge 0$。
可加性。 如果 p_1， $p_2 \in P$不相交，则 $V(p_1 \cup p_2) = V(p_1) + V(p_2)$。
连续性。 对于 ℓ， $v \in [0,1]$，如果 $v \le V([\ell,1])$，则存在 $r \in [\ell,1]$使得 $V([\ell,r]) = v$。

这些条件通常用于公平分配文献中。第一个性质将整个蛋糕的值归一化为 1。第二个性质意味着不存在“坏的”蛋糕份额。第三个性质表明，两个不相交份额的值等于各份额值之和。结合第二个性质，这意味着估值具有（弱）单调性：如果一个份额p_1 包含于另一个份额 p_2 中，那么 $V(p_1) \le V(p_2)$。

最后一条性质指出，扩大一份蛋糕时，其价值会平滑地增加，不会突然跳变。请注意，实数 r 可能并不唯一——从左端点 ℓ开始，可能存在多个价值相同的区间。尽管这条性质看起来有些复杂，但它对于支持切蛋糕协议中的基本操作至关重要。例如，它保证了对于一个价值为 v的区间 $[a,b]$，我们可以找到一个价值恰好为 $v/2$的子区间 $[a,c]$。根据可加性，该区间的其余部分（c， b]）的价值也必定为 $v/2$。换言之，连续性确保我们总能将一个区间平分为两份价值相等的份额。

连续性的另一个有用推论是原子性： $V([r,r]) = 0$对所有 $r \in [0,1]$均成立。这意味着我们可以忽略区间端点是闭的还是开的，并且可以将仅在一个端点处重叠的区间视为实际上互不相交。例如，给定 $a \le b \le c$，我们有

$$\begin{aligned} v([a,c]) &= v([a,b)) + v([b,c]) && \text{(additivity)} \\ &= v([a,b)) + v([b,b]) + v([b,c]) && \text{(atomicity)} \\ &= v([a,b]) + v([b,c]) && \text{(additivity)} \end{aligned}$$

因此，当两个区间共享一个端点时，可加性仍然成立。

分配和无嫉妒性。 一个分配是将$\{p_a\}_{a\in\mathcal{A}}$不相交的份额分配给参与者的一种指派；代理人 a 接收份额 p_a。分配中的份额并集不要求涵盖整个蛋糕——我们可以留下一部分蛋糕不进行分配。

公平分配的目标是实现一种无嫉妒 分配。

定义 *2.1*。 一个分配 $\{p_a\}_{a\in\mathcal{A}}$，如果对于所有 a， $a'\in$A， $V_a(p_a) \geq V_a(p_{a'})$，则是无嫉妒 的。

直观地说，没有任何代理人更喜欢他人的份额而不是自己的份额。无嫉妒分配的最简单示例是 $p_a = \emptyset$ 对所有 $a\in$A。这是无嫉妒的，因为 $V_a(\emptyset) = 0$ 对所有 $a\in$A，因为

$$V_a(\emptyset) = V_a(\emptyset\cup\emptyset) = V_a(\emptyset) + V_a(\emptyset).$$

虽然这种分配是无嫉妒的，但它相当浪费。公平分割协议通常旨在分配大部分蛋糕，如果不是全部的话。

2.2 切蛋糕协议

从概念上说，切蛋糕协议将代理人估值作为输入，并将分配作为输出。然而，这一简单图景并不实用——估值会为每一种可能的份额赋予一个数值，而假设参与者能够将其完整的价值函数传达给该程序是不现实的。

因此，切蛋糕算法通常仅假设能够通过称为查询 的原语操作间接访问价值函数。在操作层面，算法可以作为交互式协议执行，其中中央协调者向特定参与者发出查询，请他们根据其价值函数执行操作，并假设参与者会诚实地遵循这些指示。常见操作包括：

平分。 一名代理人将给定的份额切成两部分，并认为这两部分价值相等。
Trim. 一名代理人将一份额切成两部分，通常称为主要部分 和修剪部分，使得代理人对主要部分与给定的参考部分感到无差异。
比较。 代理人报告他们更偏好两份额中的哪一份。
选择。一名代理人从给定的份额集合中拿取自己最喜欢的份额。
然后，协议描述要求参与者按照某种指定顺序执行操作。协议可以根据查询的结果进行分支，例如，如果某位参与者报告一个份额比另一个份额大，就执行以下一个子协议；否则执行另一个子协议。最后，参与者选择的份额构成分配给该参与者的最终分配。

虽然操作模型看起来很简单，但其中有一些细微之处。首先，操作可能有前置条件——如果不满足这些要求，那么该操作就没有明确定义。例如，某个动作可能要求代理人将一份额Trim至价值为 v，但如果整份额的价值严格小于 v，这就是不可能的。其次，协议的执行可能是非确定性的。例如，如果要求一名代理人将一份额除法成两份他们认为同样好的份额，那么可能存在多种不同的除法，因为对于蛋糕的某些部分，该代理人的价值可能为零。尽管进行除法的代理人可能并不在意这个选择，但该选择可能会影响其他参与者的行为，并影响协议的其余部分。无嫉妒协议必须对所有 可能的执行都满足无嫉妒。

2.3 玩具示例：切分—选择

一种在两位参与者之间分割物品的经典协议是切分—选择：
(1)首先让代理人进行除法，将蛋糕分成两个相等的份
额。(2) 让第二位代理人选择其偏好的份额。

(3) 请第一位代理人挑选剩下的份额。

此协议可以用我们语言的简化版本编写成如下程序；第3 节将详细介绍我们的语言。

$$
\begin{array}{l}
\text{let } (\mathit{piece}, \mathit{piece}') = \text{split}_1(\text{cake}) \text{ in} \\
\text{if compare}_2(\mathit{piece}, \mathit{piece}') = \text{bigger then} \\
\quad \text{alloc}(1 \mapsto \mathit{piece}', 2 \mapsto \mathit{piece}) \\
\text{else} \\
\quad \text{alloc}(1 \mapsto \mathit{piece}, 2 \mapsto \mathit{piece}')
\end{array}
$$

上文中，操作“平分”和“比较”对应于我们在上文介绍的动作 **平分** 和**比较**。我们在这些操作上使用下标 1 和 2，以表示被要求执行该动作的是哪个代理人，也就是查询哪个代理人的估值。我们假设，比较操作在代理人认为第一份额大于第二份额时返回 bigger，否则返回 smaller。最后，alloc 通过指定哪个代理人获得哪一份额来确定最终分配。

对于这个协议，证明无嫉妒性很容易。第一位代理人最初将蛋糕平分，因此对两份份额无差异。如果第二位代理人认为份额比份额′大，那么第二位代理人会得到份额，不会产生嫉妒；同样，如果第二位代理人认为份额比份额′小，那么第二位代理人会得到份额′，同样不会产生嫉妒。因此，无论最终如何分配，两位代理人都不会产生嫉妒：切分—选择始终能产生无嫉妒的分配。

2.4 将无嫉妒性归约为约束可满足性

为了自动化无嫉妒性的证明，我们将程序翻译为一阶逻辑中的约束，并将其交由SMT 求解器处理。我们的翻译是组合式的，由程序的结构引导。从高层次来看，我们的方法分为三个步骤。

首先，我们将程序 e 转换为一阶公式 $C[e]$，其中每个代理人都有一个自由变量 v_a，用于表示代理人 a的价值函数，以及一个特殊的自由变量 α，用于表示最终分配。然后，假设 $\mathrm{A} = \{1, \ldots, n\}$，我们构造谓词 $\mathit{EnvyFree}(v_1, \ldots, v_n, \alpha)$，断言 α对于估值 $v_1, \ldots, v_n$是无嫉妒的。同样地，我们构造谓词$\mathit{ValidValn}(v_1, \ldots, v_n)$，断言这些估值满足我们在本节开头引入的假设。最后，我们将这些谓词组合成一个捕获无嫉妒性的单一约束，其中 v 表示估值变量的序列 $v_1, \ldots, v_n$:–

$$\forall \overline{v}.\, \forall \alpha.\, C[e](v, \alpha) \wedge \mathit{ValidValn}(\overline{v}) \implies \mathit{EnvyFree}(\overline{v}, \alpha). \tag{1}$$

换言之，该公式表明，对于所有估值函数 v_1、……、 v_n 以及所有可能的分配α，如果 α 是程序针对给定估值所产生的一个可能输出 e，且给定估值满足这些公理，那么分配 α 就是无嫉妒的。（此处省略了一些细节；完整细节请参见第4节。）

在我们的贯穿示例中，切分—选择协议被转换为以下约束：

$$C[\text{Cut-Choose}] = \exists \rho, \rho'.\ \underbrace{v_1(\rho) = v_1(\rho')}_{\text{平分}_1} \wedge \text{if } \underbrace{v_2(\rho) \geq v_2(\rho')}_{\text{比较}_2} \text{ then } \underbrace{\alpha = (\rho', \rho)}_{\text{分配}} \text{ else } \underbrace{\alpha = (\rho, \rho')}_{\text{分配}}.$$

存在变量 ρ 和 ρ' 分别对应程序变量 份额 和 份额′。注释描述了与不同子公式相对应的程序操作。

将上述公式与谓词 *ValidValn* 和 *EnvyFree* 结合，如（1）所示，我们便得到一个一阶逻辑公式，它表明，对于满足公理的所有估值，切分—选择都是无嫉妒的。随后可以通过自动求解器验证这一性质。

3 SLICE：公平分配语言

现在我们已经从整体上了解了系统的工作方式，接下来将定义用于公平分割协议的语言 Slice。我们首先介绍查询模型，然后给出该语言的语法和操作语义。

3.1 罗伯逊-韦伯查询模型

正如我们所讨论的，公平分配中的协议无法直接访问参与者的估值；相反，协议会要求参与者根据其个人估值执行特定操作。形式化并比较具有不同允许操作的协议的一种典型方式，是通过查询模型来实现的，该模型描述了协议可以针对参与者的估值函数进行的查询。为了在实践中实现这些查询，协议可以要求参与者回答查询或执行某项操作。

蛋糕切分领域的大量文献已逐渐采用罗伯逊-韦伯（RW）查询模型。该模型在假设查询在实践中易于实现的同时，支持丰富的协议。该模型由罗伯逊和韦伯 [1998]提出，并由 Woeginger 和 Sgall[2007], 进一步完善（并命名），该模型允许两种查询：

求值。给定区间 $[\ell, r]$和代理人 a，返回代理人 a 对区间的估值：$V_a([\ell, r])$。**标记**。给定左端点 ℓ、代理人 a和目标值 $v \in [0, 1]$，使得 $v \leq V_a([\ell, 1])$，返回 r，使得 $V_a([\ell, r]) = v$。

评估查询很直接，但标记查询则更微妙。首先，标记查询会返回单位区间中的一个点 r，直观来说，这个点是蛋糕可以切分的位置，但实际上并不会在该位置切开蛋糕。协议可以在后续计算中使用这个标记，例如比较不同的标记，看看哪个最大。其次，如果不对估值函数作出某些假设，就无法确定标记 r 是否总是存在——例如，如果估值从 跳变 $v - 0.1$ 到 $v + 0.1$，就不可能返回一个标记，使其所描述的份额价值恰好等于 v。然而，第**2**节引入的连续性假设确保不会发生这种跳变。最后请注意，标记 r 可能并非唯一：对于特定的目标价值，可能存在多个可行的标记 v。协议无法控制代理人如何选择标记，因此标记操作是非确定性的。

示例：在 RW 模型中实现操作。可以在查询模型之上实现代理人的具体行为和操作。例如，我们在第 2 节中介绍的四种操作——**除法**、**Trim**、**比较**和**选择**——都可以通过 RW 查询实现。操作 **比较** 和 **选择**可以通过执行评估查询并比较所得值来实现；它们分别要求代理人比较两份份额，或选出自己最喜欢的一份。操作**除法** 和 **Trim**则可以通过评估查询和标记查询实现。

例如，假设我们要把区间 $[\ell, r]$ 分成两份，使代理人 a认为两份同样好。首先，我们执行评估查询，得到 a对 $[\ell, r]$的估值；将这个值记为 z，并假设它不为零。然后，我们从 ℓ 开始执行标记查询，目标值为 $z/2$。由于 $z/2$ 严格小于 $[\ell, r]$ 的值，标记查询将返回一个点 $c \in [\ell, r]$，使得 $[\ell, c]$ 的值恰好为 $z/2$。由于我们假设估值满足可加性（第 2 节），剩余份额（$c, r]$ 的值也必定恰好为 $z/2$。因此，我们可以得到区间$[\ell, z]$和 $[z, r]$。

3.2 语言语法

现在，我们已经准备好介绍这门语言。Slice 是一种标准的一阶语言，并扩展了几个自定义构造。我们将简要介绍表达式和值的语法，具体如下。

$$\begin{aligned}\mathrm{e} ::= \quad & n \in \mathbb{N} \mid r \in \mathbb{R} \mid \text{真} \mid \text{假} \mid o(e_1, \ldots, e_n)\ (o \in \mathrm{O}) \mid x \in \mathrm{X} \mid \text{令}\ x = e_1 \text{于}\ e_2 \mid (e_1, \ldots, e_n) \mid \pi_n\, e \mid \text{若}\ e_1 \text{则}\ e_2 \text{否则}\ e_3 \mid \text{蛋糕} \mid \text{左}\ e \mid \text{右}\ e \mid \\ & \text{除法规则}(e_1, e_2) \mid \text{标记}_a(e_1, e_2) \mid \text{求值}_a(e)\ (a \in \mathrm{A}) v ::= \quad n \in \mathbb{N} \mid r \in \mathbb{R} \mid \text{真} \mid \\ & \text{假} \mid (v_1, \ldots, v_n) \mid [r_1, r_2]\ (r_1 \le r_2)\end{aligned}$$

图 1. 表达式和值 inSlice

图 1。常量可以是自然数、实数或布尔值。我们假设有一个固定的O 原语操作 o，例如通常的比较运算（如相等 $e_1 = e_2$ 和小于$e_1 \le e_2$）和布尔运算，以及算术运算符（如加法和乘法）。语言中的变量 x 通过 let 绑定进行绑定；我们假设变量名取自一个可数无限集合 $\mathcal{X}$。作为标准构造的补充，该语言还具有 n元积和投影，以及条件表达式。

其余表达式是切蛋糕协议所特有的。首先，我们有用于操作区间的操作，这些区间表示蛋糕块。单位区间（整个蛋糕）用 cake表示。区间的端点可以分别使用左 e 和右 e计算。操作除法(e_1, e_2)通过在位置 e_2处切割，将区间 e_1 平分为两个子区间，其中位置 e_2是一个实数。最后，我们有来自 Robertson-Webb 模型的构造：标记$_a(e_1, e_2)$是在左端点 e_1上、目标值为 e_2的标记查询，而评估$_a(e)$是在区间 e上的评估查询。这些操作由一个代理人 a索引；我们假设A 是一个固定的有限参与者集合。份额由有限个区间组成，用区间元组表示。值完全采用标准形式。区间 $[r_1, r_2]$ 由满足$r_1 \le r_2$的实数对表示。变量不能出现在值中。

我们为我们的语言赋予一个简单的类型系统，其中包含基本类型（布尔值 $\mathbb{B}$、自然数 $\mathbb{N}$、实数 $\mathbb{R}$、位置 $\mathbb{E}\,\mathrm{in}\,[0,1]$,和区间 $\mathbb{I}$）以及 n-元积；我们用 T表示类型的集合。我们省略类型系统的描述，因为它完全是标准的，并且始终假设所有程序都是类型正确的。

3.3 操作语义

我们以大步风格定义程序的语义，判断式的形式为：

$$\langle e, \sigma \rangle \Downarrow v.$$

上述，环境 $\sigma : \mathrm{X} \rightharpoonup \mathrm{V}$ 是一个将值赋给变量的偏映射；我们将表达式与环境的配对e, σ称为一个配置。我们假设变量具有类型，因此对于每种类型 $\tau \in \mathrm{T}$，都有互不相交的集合 X_τ，并将 V_τ 记作类型为 τ 的值的集合。为简化记号，在我们对形式系统的介绍中省略类型标注。

我们的操作语义由一组估值函数 $\{V_a\}_a$ 参数化，每个代理人 $a \in \mathrm{A}$对应一个估值函数，且每个估值都满足第 2 节中的条件。图 2 展示了我们的操作语义规则。大多数规则都是标准的。变量规则 [Var] 会在环境中查找变量的值。对于操作 [Ops], , 我们假定每个元数为 n 的操作$o \in \mathrm{O}$都被解释为一个从 V^n 到 V 的函数 $[\![o]\!]$；操作的类型由某个签名给出。配对 [Pair]和投影 [Proj] 都是标准的，条件表达式的规则（[IfTrue] 和 [IfFalse]）也是如此。let 绑定 [Let] 会将 e_1 求值为值 v_1，然后扩展环境，添加一个将 x 映射到 v_1的新绑定，接着对主体求值。

$$\frac{}{\langle v, \sigma\rangle \Downarrow v}\ \textsc{Val} \qquad \frac{\langle e_1, \sigma\rangle \Downarrow v_1 \quad \cdots \quad \langle e_n, \sigma\rangle \Downarrow v_n \quad [\![o]\!] : \mathcal{V}^n \to \mathcal{V}}{\langle o(e_1, \ldots, e_n), \sigma\rangle \Downarrow [\![o]\!](v_1, \ldots, v_n)}\ \textsc{Ops}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow v_1 \quad \cdots \quad \langle e_n, \sigma\rangle \Downarrow v_n}{\langle (e_1, \ldots, e_n), \sigma\rangle \Downarrow (v_1, \ldots, v_n)}\ \textsc{Pair} \qquad \frac{\langle e, \sigma\rangle \Downarrow (v_1, \ldots, v_n) \quad 1 \le k \le n}{\langle \pi_k\ e, \sigma\rangle \Downarrow v_k}\ \textsc{Proj}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow \mathsf{true} \quad \langle e_2, \sigma\rangle \Downarrow v_2}{\langle \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3, \sigma\rangle \Downarrow v_2}\ \textsc{IfTrue} \qquad \frac{\langle e_1, \sigma\rangle \Downarrow \mathsf{false} \quad \langle e_3, \sigma\rangle \Downarrow v_3}{\langle \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3, \sigma\rangle \Downarrow v_3}\ \textsc{IfFalse}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow v_1 \quad \langle e_2, \sigma[x \mapsto v_1]\rangle \Downarrow v_2}{\langle \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2, \sigma\rangle \Downarrow v_2}\ \textsc{Let} \qquad \frac{}{\langle x, \sigma\rangle \Downarrow \sigma(x)}\ \textsc{Var}$$

$$\frac{\langle e, \sigma\rangle \Downarrow [r_1, r_2]}{\langle \mathsf{left}\ e, \sigma\rangle \Downarrow r_1}\ \textsc{Lt} \qquad \frac{\langle e, \sigma\rangle \Downarrow [r_1, r_2]}{\langle \mathsf{right}\ e, \sigma\rangle \Downarrow r_2}\ \textsc{Rt} \qquad \frac{}{\langle \mathsf{cake}, \sigma\rangle \Downarrow [0, 1]}\ \textsc{Cake}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow [r_1, r_1'] \quad \langle e_2, \sigma\rangle \Downarrow r_2 \quad r_1 \le r_2 \le r_1'}{\langle \mathsf{divide}(e_1, e_2), \sigma\rangle \Downarrow ([r_1, r_2], [r_2, r_1'])}\ \textsc{Div}$$

$$\frac{\langle e_1, \sigma\rangle \Downarrow v_1 \quad \langle e_2, \sigma\rangle \Downarrow v_2 \quad V_a([v_1, r]) = v_2}{\langle \mathsf{mark}_a(e_1, e_2), \sigma\rangle \Downarrow r}\ \textsc{Mark} \qquad \frac{\langle e, \sigma\rangle \Downarrow [r, r']}{\langle \mathsf{eval}_a(e), \sigma\rangle \Downarrow V_a([r, r'])}\ \textsc{Eval}$$

图 2. Slice 的大步操作语义

其余规则用于操作区间并实现查询。 [Cake] 表示整块蛋糕，其求值结果为单位区间 $[0, 1]$； [Lt] 和 [Rt] 分别获取区间的左端点和右端点。 [Div] 会在某个位置将区间平分为一对区间；其附加条件要求平分点必须位于该区间的边界范围内。 [Mark] 会将其第一个参数求值为 $[0, 1]$, 中的一个位置，并将第二个参数求值为目标值 v。如果存在位置 r，使得 $V_a([v_1, r])$恰好等于 v，那么程序就可以返回 r。最后， [Eval] 只需计算区间的价值。

非确定性。 如我们所见，如果代理人 a 对 $[\ell, r]$ 和 $[\ell, r\prime]$, 不加区分（即，如果代理人 a对 $[r, r\prime]$的估值为零），标记查询就可能具有非确定性。因此，我们的语言无法消除 let 绑定，因为以下程序并不等价：

$$\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \quad \not\equiv \quad e_2\{e_1/x\}$$

例如，如果 e_1 具有非确定性，且主体 e_2 两次提及 x，那么左边的程序只进行一次非确定性选择，而右边的程序会进行两次非确定性选择。

错误。 一个配置 e, σ，其中 e 不会执行一步归约但不是一个值，被称为**停滞配置**，是协议错误的结果。由于我们假设程序都是类型正确的，程序陷入停滞只有两种方式。在 [除法规则], 中，如果程序试图在给定区间的边界之外进行切割，就会陷入停滞。在 [标记], 中，如果程序无法找到一个点 r使得 $[v_1, r]$ 的值为 v_2，就会陷入停滞。根据我们对估值的假设，只有当目标值 v_2严格大于 $[v_1, r]$的值时，才会发生这种情况。

让 份额 = 除法(蛋糕,标记$_1$(0, 1/2)) 在
如果 求值$_2$(π_1 份额) ≥ 求值$_2$(π_2 份额)
则(π_2 份额 π_1 份额)否则(π_1 份额 π_2 份额)

图 3. 切分—选择 inSlice

示例：编码切分—选择。 将所有内容整合起来，假设我们有两个参与者$\mathcal{A}$ = {1,2}。图 3 展示了如何在 2 节所述的切分—选择协议中使用 Slice进行表达。输出分配表示为一个元组：第一个分量分配给参与者 1，第二个分量分配给参与者 2。

4 约束生成、可靠性与完备性

现在我们已经了解了这种语言，接下来将验证有关程序的性质。首先，我们定义一个从程序 e到约束 $c(e,$ ret) 的转换，其中 ret 是一个特殊变量。直观地说，$c(e,\ v)$ 应当恰好在 e可以执行一步得到 v时成立；我们通过证明该转换的可靠性和完备性来形式化这一论断。最后，我们将无嫉妒性表示为公式 E(ret)，并通过检查对于所有可能的输出 ret，$c(e,$ ret) 是否蕴含 E(ret) 来验证这一性质。

4.1 构建约束

我们的约束转换针对一种标准的多类别一阶逻辑。

公式的语法。 我们的逻辑的签名 ($S,\ C,\ F,\ Q$) 分别包含类型、常量、函数符号和谓词。 S中的类型恰好对应类型 T 的集合；对于每个 $\tau \in$ T，我们用 S_τ表示对应于 τ的类型。常量集合 C由 $\mathbb{N}$和 {true，false}组成。函数符号集合为

$$F = O \cup \{(_,\ldots,_)_k, \pi_k \mid k = 1,2,\ldots\} \cup \{\text{ite}, \ell, r, [_,_]\} \cup \{\text{Val}_a \mid a \in \mathcal{A}\}$$

具有预期的元数（例如，ite（条件表达式）的元数为3，估值 Val$_a$ 具有元数1，等等）。最后，谓词符号 Q由 = 和 ≥组成。

我们考虑三个互不相交的逻辑变量集合：Y = {$y_1,\ y_2,\ y_3,$ .....}、X 和 {ret}。我们将逻辑项的集合称为 Term，将公式的集合称为 Form。按照惯例，逻辑项由变量、常量以及应用于其他项的函数符号构成，而公式则由一阶逻辑中的谓词、联结词和量词构造而成（¬、 ∧、 ∨、 ⇒ ∀、 ∃）。

公式的语义。我们逻辑的一种解释将排序解释为集合，将函数符号解释为函数，将谓词符号解释为谓词。我们固定以下对逻辑的解释。

定义 *4.1*。设 μ 是一种解释。若 μ 满足以下条件，则称其为适当的

(1) $\mu(S_\tau)$ = V$_\tau$对于所有 $\tau \in$T；(2) $\mu(o) = [\![o]\!]$ 对于任意 $o \in$ O；(3) $\mu((_,\ldots,_))$ 和 π分别是积的构造函数和投影，并且 μ(ite)(a,b,c) 表示：若 a为真，则为 b；否则若 a为假，则为 c；(4) $\mu(\ell)$ 和 $\mu(r)$ 分别给出区间的左端点和右端点，并且 $\mu([_,_])$ 是由实数 r_1 和 r_2 给定的区间 $[r_1, r_2]$ ；(5) μ(Val$_a$) 是任意代理集合 A 中任意 a 的价值函数。

我们用 $\mathbb{M}$ 表示适当解释的类。给定代理集合 $\mathbb{A}$ 和估值函数集合 $V = \{V_a \mid a \in \mathbb{A}\}$，我们用 $\mathbb{M}_V$ 表示 $\overline{\mathrm{Val}}_a$ 被解释为 V_a 的适当解释类，其中对所有 $a \in \mathbb{A}$ 均成立。

我们假设解释也解释所有逻辑变量：如果 y 是排序为 S_τ 的某个变量，那么 $\mu(y) \in \mu(S_\tau)$。该解释扩展到所有项；对于逻辑项 t 的解释，我们记作 $\mu(t)$。最后，一个解释决定哪些公式是有效的。我们记作 $\mu \vDash \varphi$，如果公式 φ 在该解释中成立；记作 $\vDash \varphi$，如果 $\mathbb{M}$ 中的所有解释都满足 φ；并记作 $\mathbb{M}_V \vDash \varphi$，如果 $\mathbb{M}$ 中的所有解释都 V 满足 φ。

从表达式到公式。 我们将把程序从我们的核心语言翻译为逻辑公式。为了将编程语言的语法与逻辑公式的语法区分开来，我们将使用帽号来表示某些程序语法所对应的逻辑语法。例如，我们将 $\widehat{v}$ 写作与程序值 $v \in \mathbb{V}$ 相对应的逻辑项，并将 $\widehat{x}$ 写作与程序变量 $x \in \mathbb{X}$ 相对应的逻辑变量。

我们将定义一种具有以下类型的约束转换：

$$\mathrm{G} : \mathbb{N} \times \mathsf{Exp} \to \mathbb{N} \times \mathsf{Form} \times \mathsf{Term},$$

并且我们分别将 G_Q、G_s 和 G_ρ 称为 G 的第一、第二和第三个投影。直观地说，程序表达式 e 被转换为一个逻辑项 G_ρ，该逻辑项编码了 v that e 能够转换到的值。对求值关系进行建模存在两个复杂之处。首先，一些规则（例如 [除法规则]和 [标记]）具有必须成立的侧条件；公式 G_s 会追踪这些侧条件。其次，由于求值关系具有非确定性，一个表达式可能转换到多个不同的值。我们的约束转换通过引入新的逻辑变量来建模非确定性选择；自然数输入和输出 G_Q 会追踪逻辑变量的数量，以确保新的逻辑变量是新鲜的。

图4给出了标准构造中G 的定义。我们令 $N(e) = \mathrm{G}_Q(0, e)$，并令 $N(e_1, \ldots, e_n) = N(e_1) + \cdots + N(e_n)$；直观地说，这些量计算了在翻译表达式时引入的新逻辑变量的数量。大多数分类都很直接；下面我们评论其中较为有趣的方面。对于操作，o 遍历原语操作 $\mathbb{O}$，以及诸如左、右、配对和投影之类的构造。对于条件表达式，侧条件 G_s 将守卫 e_1 的侧条件与分类讨论合取：如果守卫为真（$\mathrm{G}_\rho(k, e_1) = \mathsf{true}$），那么第一分支 e_2 的侧条件应成立；否则，第二分支 e_3 的侧条件应成立。逻辑项 G_ρ 则只是将条件表达式应用于守卫和各分支所对应的逻辑项。

图 5 给出了我们对语言中更专用构造的约束转换。对于 除法 和 标记的转换，侧条件中的最后两个合取项反映了操作规则中的侧条件 [除法规则] 和 [标记]。最后，标记 的转换引入了一个新的逻辑变量 y，其索引经过选择，保证该变量是新鲜的。这个程序构造是语言中非确定性的来源。

最后，我们定义约束 e：

$$c(k, e, t) \triangleq \mathrm{G}_s(k, e) \wedge (t = \mathrm{G}_\rho(k, e)).$$

我们将 $c(e,\ t) \triangleq c(0,\ e,\ t)$ 简写为此。如果我们对所有逻辑变量进行存在量化来闭合此公式，那么该公式就表明逻辑项 t 描述了一个 e 可以进行单步转换的值。例如，$\exists y_1, \ldots, y_{N(e)} \cdot c(e, \widehat{v})$ 表明 e 可以归约为 v。

*Ex*示例。 将图中的切分—选择程序翻译为3 并化简，我们得到 :

$$\mathsf{G}_{Q}(0,e) = 1\mathsf{G}_{s}(0,e) = r([0,1]) \geq y_1 \geq \ell([0,1])\wedge(\mathsf{Val}_1([0,y_1]) = 1/2) \wedge (\mathsf{Val}_1([0,1]) \geq y_1)\wedge ((\mathsf{Val}_2(\pi_1(\alpha)) \geq \mathsf{Val}_2(\pi_2(\alpha))) = \mathsf{true}) \vee((\mathsf{Val}_2(\pi_1(\alpha)) \geq \mathsf{Val}_2(\pi_2(\alpha))) = \mathsf{false}))\ \mathsf{G}_{\rho}(0,e) = \mathsf{ite}(\mathsf{Val}_2(\pi_1(\alpha)) \geq \mathsf{Val}_2(\pi_1(\alpha)), (\pi_2(\alpha)\ \pi_1(\alpha)), (\pi_1(\alpha)\ \pi_2(\alpha)))$$

其中， α是 $([\ell([0,1])\ y_1],\ [y_1, r([0,1])])$ 的简写，表示在 y_1处分割后得到的两块蛋糕。请注意，虽然原始表达式中包含变量 蛋糕 和 份额，但在翻译 let 绑定时，它们分别被 $[0,1]$ 和 α 替换掉了。

我们逐步分析这三个组成部分。首先，由于此示例中有一个标记查询，约束转换引入了一个逻辑变量 y_1，并且有 $N(e) = \mathsf{G}_{Q}(0,\ e) = 1$。其次，旁条件 $\mathsf{G}_{s}(0,\ e)$ 包含三个合取项：第一个来自除法表达式，第二个来自标记表达式，第三个来自条件语句。最后，项 $\mathsf{G}_{\rho}(0,\ e)$ 表示程序产生的可能分配。

$$\begin{array}{lll}
\mathsf{G}_{Q}(k,v) \triangleq k & \mathsf{G}_{s}(k,v) \triangleq \mathsf{true} & \mathsf{G}_{\rho}(k,v) \triangleq \widehat{v} \\
\mathsf{G}_{Q}(k,x) \triangleq k & \mathsf{G}_{s}(k,x) \triangleq \mathsf{true} & \mathsf{G}_{\rho}(k,x) \triangleq \widehat{x} \\
\mathsf{G}_{Q}(k,\mathsf{cake}) \triangleq k & \mathsf{G}_{s}(k,\mathsf{cake}) \triangleq \mathsf{true} & \mathsf{G}_{\rho}(k,\mathsf{cake}) \triangleq \widehat{[0,1]}
\end{array}$$

$$\begin{array}{l}
\mathsf{G}_{Q}(k,o(e_1,\ldots,e_n)) \triangleq k + N(e_1,\ldots,e_n) \\
\mathsf{G}_{s}(k,o(e_1,\ldots,e_n)) \triangleq \mathsf{G}_{s}(k,e_1) \wedge \mathsf{G}_{s}(k+N(e_1),e_2) \wedge \cdots \wedge \mathsf{G}_{s}(k+N(e_1,\ldots,e_{n-1}),e_n) \\
\mathsf{G}_{\rho}(k,o(e_1,\ldots,e_n)) \triangleq \widehat{o}(\mathsf{G}_{\rho}(k,e_1),\ldots,\mathsf{G}_{\rho}(k+N(e_1,\ldots,e_{n-1}),e_n))
\end{array}$$

$$\begin{array}{l}
\mathsf{G}_{Q}(k,\text{如果 } e_1 \text{ 则 } e_2 \text{ 否则 } e_3) \triangleq k + N(e_1,e_2,e_3) \\
\mathsf{G}_{s}(k,\text{如果 } e_1 \text{ 那么 } e_2 \text{ 否则 } e_3) \triangleq \mathsf{G}_{s}(k,e_1) \\
\qquad\qquad \wedge ((\mathsf{G}_{\rho}(k,e_1) = \mathsf{true}) \wedge \mathsf{G}_{s}(k+N(e_1),e_2)) \\
\qquad\qquad \vee ((\mathsf{G}_{\rho}(k,e_1) = \mathsf{false}) \wedge \mathsf{G}_{s}(k+N(e_1,e_2),e_3))) \\
\mathsf{G}_{\rho}(k,\text{如果 } e_1 \text{ 那么 } e_2 \text{ 否则 } e_3) \triangleq \mathsf{ite}(\mathsf{G}_{\rho}(k,e_1),\mathsf{G}_{\rho}(k+N(e_1)\ e_2),\mathsf{G}_{\rho}(k+N(e_1,e_2)\ e_3))
\end{array}$$

$$\begin{array}{l}
\mathsf{G}_{Q}(k,\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2) \triangleq k + N(e_1,e_2) \\
\mathsf{G}_{s}(k,\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2) \triangleq \mathsf{G}_{s}(k,e_1) \wedge \mathsf{G}_{s}(k+N(e_1),e_2)\{\mathsf{G}_{\rho}(k,e_1)/x\}, \\
\mathsf{G}_{\rho}(k,\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2) \triangleq \mathsf{G}_{\rho}(k+N(e_1),e_2)\{\mathsf{G}_{\rho}(k,e_1)/x\}
\end{array}$$

图 4。基本语言构造的约束转换。

4.2 可靠性

当表达式 e 是闭合的，初始环境 σ 不会影响e 的大步语义，因此约束转换只需对程序 e进行建模。不过，由于我们的语言可以

$$
\begin{aligned}
&\mathrm{G}_{\mathcal{Q}}(k,\ \text{除法}(e_1,\ e_2)) \triangleq k + N(e_1,\ e_2)\\
&\mathrm{G}_s(k, \mathsf{divide}(e_1, e_2)) \triangleq \mathrm{G}_s(k, e_1) \wedge \mathrm{G}_s(k + N(e_1), e_2)\\
&\qquad\qquad \wedge (\mathrm{G}_\rho(k + N(e_1), e_2) \geq \ell(\mathrm{G}_\rho(k, e_1)))\\
&\qquad\qquad \wedge (r(\mathrm{G}_\rho(k, e_1)) \geq \mathrm{G}_\rho(k + n(e_1), e_2))\\
&\mathrm{G}_\rho(k, \text{除法}(e_1, e_2)) \triangleq \left(\left[\ell(\mathrm{G}_\rho(k, e_1)), \mathrm{G}_\rho(k + N(e_1)\ e_2)\right], \left[\mathrm{G}_\rho(k + N(e_1)\ e_2)\ r(\mathrm{G}_\rho(k, e_1))\right]\right)
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{G}_{\mathcal{Q}}(k,\ \text{标记}_a(e_1,\ e_2)) \triangleq k + N(e_1,\ e_2)\ + 1\\
&\mathrm{G}_s(k, \mathsf{mark}_a(e_1, e_2)) \triangleq \mathrm{G}_s(k, e_1) \wedge \mathrm{G}_s(k + N(e_1), e_2)\\
&\qquad\qquad \wedge (\mathrm{Val}_a(\left[\mathrm{G}_\rho(k, e_1), y_{k+N(e_1,e_2)+1}\right]) = \mathrm{G}_\rho(k + N(e_1), e_2))\\
&\qquad\qquad \wedge (\mathrm{Val}_a(\left[\mathrm{G}_\rho(k, e_1), 1\right]) \geq \mathrm{G}_\rho(k + N(e_1), e_2))\\
&\mathrm{G}_\rho(k, \text{标记}_a(e_1, e_2)) \triangleq y_{k+N(e_1,e_2)+1}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{G}_{\mathcal{Q}}(k, \mathrm{eval}_a(e)) \triangleq k + N(e)\ \mathrm{G}_s\\
&(k, \mathrm{eval}_a(e)) \triangleq \mathrm{G}_s(k, e)\ \mathrm{G}_\rho(k,\\
&\mathrm{eval}_a(e)) \triangleq \mathrm{Val}_a(\mathrm{G}_\rho(k, e))
\end{aligned}
$$

图5. 切蛋糕操作的约束转换。

通过 let 绑定引入局部变量——由于我们的语义是非确定性的，这些变量无法消除——因此，组合式的约束转换还必须处理开放表达式。为了陈述（并证明）可靠性定理，我们将程序环境 σ 转换为逻辑替换；形式化这一衔接是确立可靠性的主要挑战。

环境。 在本节中，将环境表示为赋值序列会很有用。设 $x_1, \ldots, x_n \in \mathrm{X}$ 为程序变量列表，这些变量不必互不相同。

定义*4.2*。 我们称 σ 为定义在 $x_1, \ldots, x_n$ 上的环境，如果 σ 是一个有限序列 $[x_1 \mapsto v_1][x_2 \mapsto v_2] \cdots [x_n \mapsto v_n]$，其中某些 $v_1, \ldots, v_n \in$ 属于 V。 σ 的定义域 为 $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$

一个环境确定一个偏映射，即， $\sigma\colon \mathrm{X} \rightharpoonup \mathrm{V}$，其中 $\sigma(x) = v_i$ 当 $x = x_i$，且 i 是满足 $x = x_i$ 的最大 i ；否则未定义。

替换。 环境将程序变量映射到值。在逻辑方面，我们可以将这些数据反映为用逻辑项替换逻辑变量。令 $x_1, \ldots, x_n \in \mathrm{X}$，同样不一定是唯一的。

定义 *4.3*。 我们称 S是一个在.....上的替换 x_1，.....， x_n，如果 $S = \{t_n/\widehat{x_n}\} \cdots \{t_1/\widehat{x_1}\}$ 对于 t_1，.....， $t_n \in$项，使得 $\mathrm{FV}(t_i) \cap \mathrm{FV}(t_j) \subseteq \mathrm{X}$ 当 $i \neq j$，且 $\mathrm{FV}(t_i) \subseteq \{x_1$，.....， $x_{i-1}\}$ 对于 $i = 1$，.....， n。

S的定义域 是 $\mathrm{dom}(S) = \{x_1, \ldots, x_n\}$。我们令 $m(S)$ 表示满足以下条件的最大自然数：y_k是 $t_1, \ldots,$ t_n之一中的自由变量。

替换可以应用于逻辑公式和项。回顾替换的常用记法，我们将 φS 记作将 S 应用于 φ 后得到的公式，将 tS 记作将 S 应用于 t后得到的项。替换以通常的方式在语法上定义；替换列表从左到右按顺序应用。

我们可以通过解释将替换与环境联系起来。

定义 *4.4*。令 μ是一个适当的解释。我们称 μ结合 σ与 S，若 $\mu(\widehat{x}S) = \sigma(x)$ 对所有 $x \in \mathrm{dom}(\sigma) = \mathrm{dom}(S)$均成立。

为了陈述我们翻译的可靠性，我们需要用到两种记号。首先，若两个解释 μ和 μ'在以下范围内一致：k，我们就称它们一致于 $\{y_1、\ldots\ldots、y_k\}$。其次，若 $V = \{V_a \mid a \in \overline{\mathrm{A}}\}$ 是一个估值函数集合，那么我们记作 $e,\ \sigma \Downarrow_{\underline{V}} v$，表示当参与者的估值为 V 时，配置 $e,\ \sigma$ 可以规约到 v。

定理 *4.5*（可靠性）。设 e 是一个类型良好的表达式，σ 是一个环境，满足 $FV(e) \subseteq dom(\sigma)$；$S$ 是一个替换，V 是一个估值函数集合，$v \in$ 是 V，μ 是一个来自 M_V 的解释，它将 σ 与 S 配对。若 $\langle e, \sigma\rangle \Downarrow_V v$，则对于任意 $k \geq m(S)$，存在一个来自 M_V 的解释 μ'，它在 $\{y_1, \ldots, y_k\}$ 上与 μ 一致，且 $\mu' \vDash c(k, e, \widehat{v})S$。

关于定理4.5的证明，请参见附录A。我们可以得出闭表达式的可靠性。

推论4.6。令 e 为一个闭表达式。令 V 为一个估值集合，且令 v 为一个值。如果$\langle e, \varepsilon\rangle \Downarrow_V v$，那么$\mathrm{M}_V \vDash \exists y_1、\ldots、y_{N(e)}.c(e, \widehat{v})$。

4.3 完备性

可靠性表明，如果程序 e 可以规约为 v，那么当我们代入 v时，我们的约束是可满足的。反之亦然。

定理 4.7（完备性）。设 e 是一个类型正确的表达式，S 是一个替换，满足$\mathrm{FV}(e) \subseteq dom(S)$，$V$ 是一组估值函数，$v \in \mathcal{V}$，且 σ 是一个环境。若存在 $k \geq 0$ 和一个从M_V出发、将 σ与 S结合起来的解释 μ，使得 $\mu \vDash c(k, e, \widehat{v})S$，则$\langle e, \sigma\rangle \Downarrow_V v$。

关于定理4.7的证明，请参见附录 A。由此可得，闭表达式的完备性是一个特殊情况。

推论 4.8。设 e 是一个没有自由变量的类型正确的表达式，V 是一组估值函数，并且 $v \in \mathcal{V}$。如果 $\mathrm{M}_V \vDash \exists y_1、\ldots\ldots、y_{N(e)}.c(e、\widehat{v})$，那么 $\langle e、\varepsilon\rangle \Downarrow_V v$。

因此，$e, \varepsilon \Downarrow_V v$当且仅当存在 $\mu \in \mathrm{M}_V$ 使得 $\mu \vDash c(e, \widehat{v})$，等价地，$\mathrm{M}_V \vDash \exists y_1, \ldots, y_{N(e)}.c(e, \widehat{v})$。因此，$e$的约束完全刻画了其输出。

4.4 验证无嫉妒性

有了约束转换，我们就能够构造约束来确保无嫉妒性等性质。设 e 是一个没有自由变量的表达式，$\varphi(\mathrm{ret})$ 是一个将 ret 作为自由变量的公式，其中 ret 的排序与 e 的类型一致。若对于所有估值集合 V 和值 $v \in \mathrm{V}$，我们都有，则记为 $e \vDash \varphi(\mathrm{ret})$。

$$\langle e, \varepsilon\rangle \Downarrow_{\overline{V}} v \text{ implies } \mathrm{M}_{\overline{V}} \vDash \varphi(\widehat{v}),$$

其中 $\varphi(\widehat{v}) = \varphi(\mathrm{ret})\{\widehat{v}/\mathrm{ret}\}$。

现在，设 $\mathrm{Val}_a^k(p) \triangleq \mathrm{Val}_a(\pi_1 p) + \cdots + \mathrm{Val}_a(\pi_k p)$，其中 t的排序为 $S_{\mathbb{E}}^k$，这是一个表示代理人 a 对份额 p估值的公式，该份额由 k 个区间组成。假设 ret 表示程序计算出的分配，而 k_a 是分配给代理人 a的区间数。设

$$E_a(\mathrm{ret}) \triangleq (\mathrm{Val}_a^{k_a}(\pi_a\,\mathrm{ret}) \geq \mathrm{Val}_a^{k_1}(\pi_1\,\mathrm{ret})) \wedge \cdots \wedge (\mathrm{Val}_a^{k_a}(\pi_a\,\mathrm{ret}) \geq \mathrm{Val}_a^{k_n}(\pi_n\,\mathrm{ret})),$$

然后，以下公式说明 ret 是一种无嫉妒分配：

$$E(\mathrm{ret}) \triangleq \bigwedge_{a \in \mathcal{A}} E_a(\mathrm{ret}).$$

不难看出，如果分配 v 满足 $E(\widehat{v})$，对于代理人估值 V 而言，那么该分配相对于 V 是无嫉妒的。因此，如果 $e \vDash E(\text{ret})$， e 是无嫉妒的。

现在，我们可以利用可靠性和完备性。首先，我们考虑公式

$$\psi \triangleq \forall \text{ret}.(\exists y_1, \ldots, y_{N(e)}.c(e, \text{ret})) \Rightarrow E(\text{ret}).$$

现在，对于任意一组估值 V 以及任意满足 $v \in \mathcal{V}$ 使得 $\langle e, \varepsilon\rangle \Downarrow_V v$，根据推论 4.6，存在一个 M_V 中的解释，满足 $c(e, \widehat{v})$，因此 $\mathrm{M}_V \vDash \exists y_1, \ldots, y_{N(e)}.c(e, \widehat{v})$。所以，如果 $\vDash \psi$，那么 $\mathrm{M}_V \vDash E(\widehat{v})$。如果 $\nvDash \psi$，那么存在某组估值 V、一个解释 $\mu \in \mathrm{M}_V$，以及一个值 v，使得

$$\mu \vDash \exists y_1, \ldots, y_{N(e)}.c(e, \widehat{v}) \wedge \neg E(\widehat{v}).$$

根据定理 4.7，$\langle e, \varepsilon\rangle \Downarrow_V v$ 因此 $e \nvDash E(\text{返回})$。因此， $e \vDash E(\text{返回})$ 当且仅当 $\vDash \psi$。正如我们将在第 6 节讨论的那样，我们可以使用 SMT 求解器来检查 $\vDash \psi$。

5 Slice 中的公平分配协议

为了评估 Slice，我们实现了公平分配文献中的若干协议。

5.1 记号、约定和语法糖。

公平分配协议通常使用大量的分支、操作和临时变量。为了让我们的示例程序更易于阅读，我们采用了几项约定。

元变量和记号。 我们使用大写字母（通常是 I或 A、 B.....）表示取值于代表蛋糕块的区间的变量。我们使用 v 表示取值于值的变量， m 表示取值于标记（即单位区间上的位置 $[0, 1]$）的变量，以及 a 表示取值于参与者的变量。程序使用数字（1、2、3、..... ）来指定参与者，有时也简称为 P1、P2、P3等。为了减少变量的数量，我们的程序经常对变量进行遮蔽/重新绑定。与第 3节一样，诸如 标记a 和 估值a等操作的下标表示应使用哪个参与者的估值。

语法糖。我们使用标准语法糖，例如用于 let 绑定一个配对：

$$\text{let } (I, J) = e \text{ in } e' \triangleq \text{let } H = e \text{ in let } I = \pi_1\, H \text{ in let } J = \pi_2\, H \text{ in } e' \qquad (H \text{ fresh}).$$

请注意，我们必须先对表达式进行 let 绑定 e 再进行投影，这样两个组成部分的非确定性选择就只会做出一次。我们使用关键字 alloc 来表示最终的分配。例如， $\text{alloc}(A, B)$ 表示将 A 分配给代理人 1 ，并将 B 分配给代理人 2 的分配。这个关键字并不存在于我们的核心语言中（第 3 节）——它仅仅是一条注释，用于突出显示最终的分配。

用户定义的缩写。为减少代码重复，我们将使用基本的用户定义的缩写/宏。例如，以下缩写接受两份额 I_1 和 I_2，并根据代理人 a的估值按降序排列：

$$\begin{aligned}&\text{def sort}_a(I_1, I_2) = \\ &\quad \text{let } v_1 = \text{eval}_a(I_1) \text{ in} \\ &\quad \text{let } v_2 = \text{eval}_a(I_2) \text{ in} \\ &\quad \text{if } v_1 \geq v_2 \text{ then } (I_1, I_2) \text{ else } (I_2, I_1)\end{aligned}$$

我们有此运算的类似 n 元版本，用于对 n 份额排序。

5.2 切分—选择

先来热身一下，回顾第 3 节中的切分—选择协议。为方便起见，我们在图 6 中重现图 3中的代码，此处使用缩写 排序$_2$ 来比较各份额。这段代码实现了以下协议：

(1) P1 在 $[0,1]$ 中标记位置 m，在他们看来，标记左侧和右侧的份额同样好（价值各为 1/2 ）。(2) P1 在 m 处切分蛋糕，将其分成两份： I_1 和 I_2。(3) P2 根据自己的估值为两份排序，其中 A 是最喜欢的一份， B是另一份。(4) P1 和 P2 分别获得 B 和 A。

我们可以借助图 7中的示意图来直观呈现这一协议。P1 先切分，然后将标记为 B 和 A 的两份分配给两位参与者。示意图没有显示 P2 对份额进行排序，而且它只呈现了一种可能的分配方式——根据 P2 的估值，他们可能更喜欢左边的份额，而不是右边的份额。不过，这幅图可以粗略地说明由哪位参与者切分，以及哪位参与者获得哪一份。虽然用文字解释切分—选择协议已经足够简单，但这些示意图将有助于我们直观理解接下来更复杂的协议。

切分—选择：令 $m = \text{mark}_1(0, 1/2)$，再令 $(I_1, I_2) = \text{divide}(\text{cake } m)$，然后令 $(A, B) = \text{sort}_2(I_1, I_2)$，最后 $\text{alloc}(B, A)$

图 6. 切分—选择 inSlice。

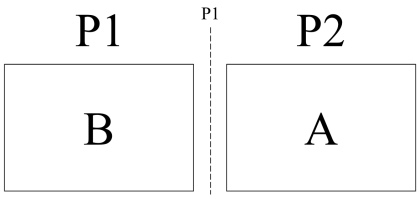


图 7. 切分—选择可能产生的分配。

5.3 剩余

在某些情形下，例如蛋糕的一部分会给双方参与者带来负效用，或者节约资源能产生社会效益，留下一部分蛋糕不予分配可能更可取。不需要分配整块蛋糕的协议称为采用自由处置 模型。当然，协议应尽可能多地分配蛋糕——把整块蛋糕都留着不分配的协议并没有多大用处。

剩余协议 [Brams 等人 2006] 对蛋糕进行划分，使两位参与者各自获得一个连通部分，并且他们认为该部分的价值至少相当于整块蛋糕总价值的一半，同时也可能留下一份未分配的“剩余”。具体步骤如下：

(1) 每位代理人标记一个位置，使其相信两份份额受到同等偏好；由于两位参与者可能有不同的估值，参与者可以在不同的位置进行标记。(2) 参与者比较标记。标记位置最靠左的代理人获得其标记左侧的份额。另一位代理人获得其标记右侧的份额。(3) 标记之间的“剩余”份额被弃置（不予分配）。

图 8 展示了我们对剩余的实现，图 9展示了一种可能的分配。这个协议的一个显著特点是，它会比较标记，以确定哪个标记最靠左

的位置——这一操作超出了第 2 节中的简单基本操作，需要使用 Robertson-Webb 模型中的查询（第3 节）。

剩余：令 $m_1 =$ 标记$_1(0, 1/2)$，再令 $m_2 =$ 标记$_2(0, 1/2)$，再令 $(A, S) =$ 对蛋糕 m_1进行除法，然后令 $(T, B) =$ 对蛋糕 m_2进行除法，若 $m_1 \geq m_2$ 则分配(S, T)，否则分配(A, B)

图 8. Slice 中的剩余。

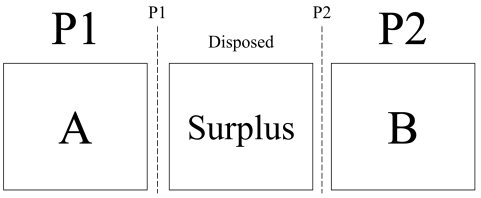


图 9. 剩余的一种可能分配。

5.4 Selfridge-Conway

我们介绍过的协议适用于 $n = 2$ 名参与者。参与者人数增加到 $n = 3$ 名后，问题会变得困难得多。针对三名参与者的首个公平分配协议由 Selfridge 提出，Conway 则独立提出了同一协议 [Robertson 和 Webb 1998]；这一协议也可作为更高级协议的构建模块。从整体上看，该协议的步骤如下：

(1) P1将蛋糕平分成三份，他们认为每份相等（价值恰好为 1/3）。（2）P2 将他们三份中最喜欢的那份分成 *Trim* 与剩余部分，使*Trim* 与他们第二喜欢的份额相等。（3）修剪部分*Trim* 和另外两份未修剪的份额由参与者按 P3、P2、P1 的顺序进行选择，但须遵守以下约束：如果P3 不选择*Trim*，那么 P2 必须选择*Trim*。（4）如果 P2 进行了平凡切割（剩余部分 为空），协议到此结束。

接下来，对额外的份额剩余部分 执行除法。在这一步，区分是 P2 还是P3 收到了*Trim*非常重要。在 P2 和 P3中，将收到*Trim*的代理人称为 PA，另一位代理人为 PB。

(5) PB 将剩下的部分切成三份，认为它们大小相等。(6) 三位参与者按照 PA、P1、PB 的顺序选择这三份。

图10 展示了该协议的实现，以及一种可能的分配。为了让程序更易读，我们定义了一个缩写 allocRest，用于分配份额 *Rest*；请注意，它将两位代理人 PA 和 PB 以及待分配的剩余份额作为参数。

Selfridge-Conway 也可以通过直接丢弃剩余份额而非将其分割，改造成一种自由处置协议。我们称这一变体为 Selfridge-Conway-剩余协议；详情见附录D。

5.5 Waste-Makes-Haste

Selfridge-Conway 协议可以实现无嫉妒分配，但某位代理人获得的分配可能由两个不连通的部分组成。这可能并不理想，例如，当参与者分配到一段路边地带来建造商店时。Segal-Halevi 等人 [2016] 在自由处置模型中提出了一种名为 Waste-Makes-Haste 的三人协议，其中每位参与者都获得一连续的份额。该协议的第一阶段与 Selfridge-Conway 协议相同，但 P2 修剪其最喜欢的份额时采用的方式不同。随后，每位参与者按

规定的顺序，在一轮中选择自己最喜欢的份额，而不是分两轮选择两份。图11展示了该协议实现的示意图，以及一种可能的分配方案。

6 实现与评估

6.1 系统概述

我们使用 ZSlice 在 OCaml 中开发了一个原型实现，采用3 [de Moura 和 Bjørner2008] 以及 CVC5 [Barbosa 等人 2022] 的方法进行约束求解。我们的实现约有 2000 行代码。从较高层面来看，我们的工具会将以领域特定语言编写的公平分配协议转换为逻辑约束，以编码无嫉妒性和无错误性等属性，然后将这些约束交由 Z3 和 CVC5 求解。我们还为该语言实现了一个求值器，使用户能够在具体估值上运行这些经过验证的协议。

前端。表层语言与第3节所述的核心语言相当相似，并加入了一些语法糖和便于编写常见子程序的缩写功能，例如按偏好降序对份额排序。我们的实现还包含一个简单的类型检查器。

求值器。我们的求值器实现了图2中的操作语义。程序的行为取决于代理人估值，求值器具有以下类型签名：

$$\texttt{evaluate} : \texttt{Protocol} \to \texttt{Valuation list} \to \texttt{Mark list} \to \texttt{Alloc}$$

除了要执行的协议外，求值器还接受两个参数：估值函数列表和标记函数列表。这两个参数分别决定 eval 和 mark 查询的行为。虽然只提供价值函数会更简单，但如果不对价值函数的形式作进一步假设，就无法从价值函数推导出标记函数。我们的求值器不依赖估值和标记的具体实现方式。实际上，完整的估值函数和标记函数甚至可能不会被显式提供；协议的执行过程可能会涉及询问实际的参与者，了解他们如何评估或标记给定份额。最后，求值器会返回一个分配：由区间元组构成的元组，用于说明每位参与者获得了哪些区间。

估值函数。我们的实现将估值视为从区间到数值的函数。这并非一种限制，因为协议只进行有限次切分，因此每位参与者只会得到有限个互不相交的区间，并且假设估值具有可加性。附录中列出了用于估值的求解器公理B。

约束生成。 我们的实现会将抽象语法转换为逻辑公式，然后将其发送给 Z3 和 CVC5。我们将估值建模为从 $\mathbb{R}^2$ 到 $\mathbb{R}$ 的函数，并将关于估值的假设编码为逻辑公式；当估值应用于两个参数 (l, r) 且 $l \le r$ 时，这些公式成立。

验证程序性质。 正如第 4 节末尾所讨论的，我们可以证明协议 e 满足某个性质 $\varphi(\text{ret})$ 通过证明

$$\forall\, \text{ret}\,.(\exists y_1, \ldots, y_{N(e)}.c(e, \text{ret})) \Rightarrow \varphi(\text{ret})$$

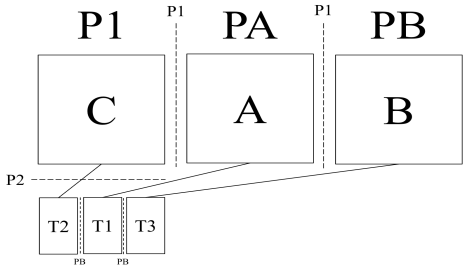
是有效的，或者等价地，

$$\exists\, \text{ret}\,.(\exists y_1, \ldots, y_{N(e)}.c(e, \text{ret})) \wedge \neg\varphi(\text{ret}) \tag{2}$$

不可满足。一旦我们生成协议的约束，就将（2）传递给Z3和 CVC5，以检查其可满足性。我们将此技术应用于验证无嫉妒性，针对第4节末尾表示无嫉妒性的公式。

```
塞尔弗里奇-康威法：
  令 (I1, I′1) = 除法规则(蛋糕, 标记1(0, 1/3)) in
  令 (I2, I3) = 除法(I′1, 标记1(0, 2/3)) 中
  令 (A, B, C) = 排序2(I1, I2, I3) 中
  如果 eval2(A) = eval2(B) 那么
    let (A, B, C) = 排序3(A, B, C) in
    let (B, C) = 排序2(B, C) in
    alloc(C, B, A)
  else
    let m = 标记2(左 A, eval2(B)) in
    let (Trim, Rest) = 除法(A, m) in
    let (A, B, C) = 排序3(Trim B, C) in
    if A ≠ Trim then
      let (R1, R2, R3) = allocRest(2, 3, 剩余) in
      如果 B = Trim 那么
        alloc((C, R1), (B, R2), (A, R3))
      else
        alloc((B, R1), (C, R2), (A, R3))
    else
      let (B, C) = 排序2(B, C) in
      let (R1, R2, R3) = allocRest(3, 2, Rest) in
      alloc((C, R1), (B, R3), (A, R2))

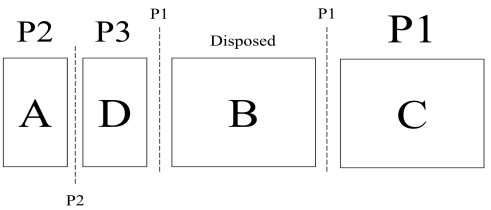
def allocRest(PA, PB, Rest) =
  let v = evalPB(剩余) in
  let m1 = 标记PB(左 剩余 v/3) in
  令 (I1, I′1) = 除法(剩余部分 m1) 中
  令 m2 = 标记PB(左 I′1, (2/3) ∗ v) 在
  令 (I2, I3) = 除法(I′1, m2) 于
  令 (I1, I2, I3) = 排序PA(I1, I2, I3) 中
  let (I2, I3) = 排序1(I2, I3) in
  alloc(I2, I1, I3)
```



Selfridge-Conway 的一种可能分配

图 10. Selfridge-Conway inSlice

```
浪费促成匆忙：
  let (I1, I′1) = 除法(蛋糕, 标记1(0, 1/3)) in
  设 (I2, I3) = 除法(I′1, 标记1(0, 2/3)) 中
  令 (A, B, C) = 排序2(I1, I2, I3) 在
  如果 eval2(A) = eval2(B) 那么
    let (A, B, C) = 排序3(A, B, C) in
    令 (B, C) = 排序2(B, C) 中
    alloc(C, B, A)
  else
    let (Trim D) =
      if (eval2(A) ≥ 2 ∗ eval2(B)) then
        除法(A, 标记2(0, (1/2) ∗ eval2(A))
      else
        除法(A, 标记2(左 A, eval2(B)))
    in
    令 (A, B, C, D) = 排序3(Trim B, C, D) 于
    如果 A = Trim 那么
      令 (B, C, D) = 排序2(B, C, D) 在
      let (C, D) = 排序1(C, D) in
      alloc(C, B, A)
    else
      令 (B, C, D) = 排序1(B, C, D)
      如果 B = Trim 则
        分配（C, Trim A）
      else
        alloc(B, Trim A)
```



来自 Waste-Makes-Haste 的可能分配

图 11. Waste-Makes-Haste 的 inSlice

表 1. 验证无嫉妒性（5 次运行的平均计时）

	大小（行数）		计时（秒）		
协议	**程序**	**约束编译**		**Z3 CVC5**	
切分—选择	5	122	0.039	0.018	0.026
剩余	5	132	0.042	0.020	失败
Waste-Makes-Haste	18	418	0.054	0.841	0.336
Selfridge-Conway-剩余协议	20	401	0.051	0.819	0.573
Selfridge-Conway-Full	22	786	0.103	19.376	53.46

6.2 评估

我们实现了第 5 节所述的协议，并使用我们的实现来验证无嫉妒性。表 1 展示了基准测试的基本指标，以及每个基准测试在五次试验中验证所需时间的平均值。实验在一台配备 4 核英特尔酷睿 i5 CPU（4.5 GHz）和 16GB 内存、运行Linux 的计算机上进行。我们尚未探索如何优化工具，因此性能很可能还可以进一步提升，但结果表明，我们的工具能够在合理的时间内验证公平分配协议。

虽然很难衡量生成的约束的规模，但我们的粗略指标也表明，对于更高级的协议，检查无嫉妒性的复杂度会迅速增加。最复杂的协议，即完整版本的 Selfridge-Conway 协议，在 Z3 和 CVC5 下进行检查所需的时间都明显长于其他协议。我们还不能完全理解为什么该协议耗时更长，但我们认为这是由于通过该协议的路径数量所致。每条路径都对应于约束中的一个不同析取项。计算我们的协议中的路径数量后，我们发现 Selfridge-Conway 协议有1,800 条路径，而 Selfridge-Conway-剩余和 Waste-Makes-Haste 的路径数量次之，均为 216 条。

7 相关工作

其他查询模型。 我们的工作聚焦于 Robertson-Webb 查询模型中的协议。尽管这一模型在公平分配文献中相当常见，但仍有一些协议需要使用其他查询模型。例如，有些协议要求参与者让多把刀平滑地在蛋糕上移动，直到某位参与者决定进行切分 [Brânzei 和 Nisan 2017]。我们认为，只需进行少量修改，我们的技术发展就可以扩展到其他查询模型；不过，考虑一种能够在不同模型之间轻松切换的更通用框架，或许会很有意思。

另一个近期模型是广义切分—选择模型 [Brânzei 等人提出的模型 2016], ，该模型允许参与者在蛋糕上进行新的切分，或选择切分之间的单个份额保留。该模型更适用于从博弈论角度看待蛋糕分配的情形，即参与者可能偏离协议。虽然支持新的查询相当直接，但当参与者可能偏离协议时，要推理其正确性则更具挑战性——无嫉妒性等性质不会无条件成立，而可能需要对参与者的理性作出假设。将此类假设纳入程序验证，是未来一个非常有吸引力的研究方向。

研究人员还研究了其他限制协议复杂度的模型。例如，在Balkanski 等人 [2014], 提出的同时模型中，协议要求每位代理人报告其估值的压缩版本，并且不允许进行任何进一步的查询。这些模型有助于研究公平分割协议的复杂度特性。

虽然我们的重点是验证无嫉妒性，但如果能探讨程序验证中的一些思路（例如资源分析）是否适用，也会很有意思。

经验证的机制设计。 形式化验证领域中，有一个规模虽小但正在发展的方向，旨在验证机制设计文献中的程序。 Barthe 等人 [2015]开发了一种用于关系属性的类型系统，并展示了如何使用该系统验证拍卖的激励属性。Barthe 等人 [2016]随后将该方法扩展到随机化机制的贝叶斯激励兼容性验证。贝叶斯激励兼容性是激励兼容性的一种更复杂的形式。由于无嫉妒性似乎并非关系属性，因此尚不清楚如何使用现有方法分析我们的目标协议。

尽管我们的工作旨在实现完全自动化的验证，研究人员已经在交互式定理证明器中形式化了各种经济机制和博弈论性质。例如，Bai 等人 [2013]；Caminati 等人 [2015] 在 Coq中形式化了 VCG 机制的激励性质，而开发了一个用于算法博弈论的 Coq 库 [Bagnall 等人 2017]并机械化证明了某些在线学习算法会收敛到近似均衡 [Merten 等人2018]。

求解器辅助编程。 最后，在编程语言方面，我们的工作属于求解器辅助编程语言这一广义范畴。这类编程语言将程序转换为逻辑公式，然后将目标验证或综合目标表示为逻辑约束，再由自动化 SMT 求解器处理。该领域的一些代表性工具包括 Dafny [Leino 2010], F* [Swamy 等人 2013], 和 Rosette [Torlak 和 Bodík 2013]。虽然可以使用其中一个系统来实现我们的方法，但目前尚不清楚如何处理诸如标记之类的非确定性查询。我们的协议和性质也是高阶的，因为协议依赖于未知的价值函数，并且假设这些价值函数满足复杂的量化公理。目前尚不清楚如何将我们的协议和性质编码到现有的求解器辅助工具中。此外，由于我们的程序相当简单，我们不需要求解器辅助编程语言的全部通用性。通过开发我们自己的、更受限制的语言，我们能够证明约束生成过程的可靠性和完备性，并且能够更好地针对目标应用定制约束生成过程。

8 结论与未来展望

在这项工作中，我们开发了一种用于公平分配协议的核心语言，并提出了一种约束生成过程，用于编码无嫉妒性和无错误性属性。我们的原型实现 Slice 表明，可以自动编码并检查这些协议的属性。我们认为未来有几个自然的研究方向。

验证不相交性。无嫉妒性是公平分配中典型的目标属性，但还有许多其他并不总是显而易见的正确性属性。例如，协议应始终产生一个分配：将整个蛋糕划分为互不相交的部分；如果允许自由处置，则也可以是蛋糕某个子集的不相交划分。开发一种线性类型系统 [Girard 1987] 以确保蛋糕的任何部分不会被分配超过一次，可能会很有趣；仿射类型系统或许有助于允许自由处置。

开发定制求解器。迄今为止，我们一直依赖通用 SMT 求解器（Z3 和 CVC5）来求解约束。然而，我们的协议所需的约束高度针对公平分配，因此，将约束生成面向更专门的求解器，甚至为这些约束开发定制求解器，可能会很有意义。例如，关于集合约束求解器的既有研究（如 [Aiken 1999]）或许适用于我们的场景。

公平分配的组合子。最后，虽然我们已实现协议验证自动化，但仍存在一个主要瓶颈：首先要将协议实现出来。将现有的高级公平分配协议介绍（例如，适用于四名或更多参与者的协议）转换为我们的语言，极其耗时，往往还需要专家知识来消除歧义，并根据英文描述展开协议。若能提取构建公平分配协议的模式，将会非常有用；我们可以设想使用这些组合子来组织算法，从而更轻松地描述和实现协议、减少代码重复，并改进约束生成。通过初步实验，我们已发现一些可能的组合子，但我们认为仍有许多工作有待探索。

实现访问

我们已在线提供了我们的原型实现 [Bertram 等人 2023]。其中包括实现源代码、第5节示例的实现，以及帮助复现表1 并使用我们的工具实现其他协议的脚本和分步说明。

致谢

我们感谢匿名评审和负责指导论文修改的审稿人认真审阅并提出宝贵建议。本研究受益于康奈尔大学编程语言讨论组（PLDG）多轮反馈。本研究部分由康奈尔大学和美国国家科学基金会（资助编号 # 1943130）支持。

参考文献

Alexander Aiken. 1999. 集合约束式程序分析导论。《计算机程序科学》35, 2 (1999), 79–111. https://doi.org/10.1016/S0167-6423(99)00007-6 Haris Aziz 和 Simon Mackenzie. 2016a. 适用于任意数量参与者的离散且有界无嫉妒切蛋糕协议。载于 IEEE 计算机科学基础研讨会（FOCS），新泽西州新不伦瑞克。416–427. https: //doi.org/10.1109/FOCS.2016.52 Haris Aziz 和 Simon Mackenzie. 2016b. 适用于四名参与者的离散且有界无嫉妒切蛋糕协议。载于 ACM SIGACT 计算理论研讨会（STOC），马萨诸塞州剑桥。454–464. https://doi.org/10.1145/2897518. 2897522 Alexander Bagnall、Samuel Merten 和 Gordon Stewart. 2017. Ssreflect/Coq 中算法博弈论的一个程序库。《形式化推理杂志》10, 1 (2017), 67–95. https://doi.org/10.6092/issn.1972-5787/7235 Wei Bai、Emmanuel M. Tadjouddine、Terry R. Payne 和 Sheng-Uei Guan. 2013. 一种用于证书拍卖机制的携证明代码方法。载于形式化组件软件研讨会（FACS），中国南昌（《计算机科学讲义》，第 8348 卷），José Luiz Fiadeiro、Zhiming Liu 和 Jinyun Xue（编）。Springer，23–40. https://doi.org/10. 1007/978-3-319-07602-7 4_Eric Balkanski、Simina Brânzei、David Kurokawa 和 Ariel Procaccia. 2014. 同时切蛋糕。人工智能 AAAI 会议，加拿大魁北克市 28, 1 (2014 年 6 月)。https://doi.org/10.1609/aaai.v28i1.8802 Haniel Barbosa、Clark W. Barrett、Martin Brain、Gereon Kremer、Hanna Lachnitt、Makai Mann、Abdalrhman Mohamed、Mudathir Mohamed、Aina Niemetz、Andres Nötzli、Alex Ozdemir、Mathias Preiner、Andrew Reynolds、Ying Sheng、Cesare Tinelli 和 Yoni Zohar. 2022. cvc5：一款功能多样、具备工业级实力的 SMT 求解器。载于国际系统构造与分析工具及算法会议（TACAS），德国慕尼黑（《计算机科学讲义》，第 13243 卷），Dana Fisman 和 Grigore Rosu（编）。Springer，415–442. https://doi.org/10.1007/978-3-030-99524-9_24 Gilles Barthe、Marco Gaboardi、Emilio Jesús Gallego Arias、Justin Hsu、Aaron Roth 和 Pierre-Yves Strub. 2015. 用于机制设计和差分隐私的高阶近似关系精化类型。载于 ACM SIGPLAN–SIGACT 编程语言原理研讨会（POPL），印度孟买。55–68. https://doi.org/10.1145/2676726. 2677000 arXiv:1407.6845 [cs.PL]Gilles Barthe、Marco Gaboardi、Emilio Jesús Gallego Arias、Justin Hsu、Aaron Roth 和 Pierre-Yves Strub. 2016. 机制设计中的计算机辅助验证。载于网络与互联网经济会议（WINE），魁北克省蒙特利尔（《计算机科学讲义》，第 10123 卷）。Springer-Verlag，273–293. https://doi.org/10.1007/978-3-662-54110-4_20 arXiv:1502.04052 [cs.GT]Noah Bertram、Alex Levinson 和 Justin Hsu. 2023. 切蛋糕：一种用于公平分配的语言。https://doi.org/10.5281/zenodo.7806738

Steven J Brams、Michael A Jones 和 Christian Klamler. 2006. 更好的切蛋糕方法。《美国数学学会通告》53, 11 (2006), 1314–1321.

Simina Brânzei、Ruta Mehta 和 Noam Nisan. 2016. 公平分配的算法复杂性。载于 *国际人工智能联合会议（IJCAI）*。美国纽约州纽约市。https://doi.org/10.1609/aaai.v30i1.10042

SiminaBrânzei 和 Noam Nisan。2017 年。切蛋糕的查询复杂度。 https://doi.org/10.48550/ARXIV.1705.02946 Marco B Caminati、Manfred Kerber、ChristophLange 和 ColinRowat。2015 年。可靠的拍卖规范与实现。

发表于*ACM SIGecom 经济学与计算会议（EC）*，俄勒冈州波特兰。547–564。https://doi.org/10.1145/ 2764468.2764511

Leonardo Mendonça de Moura 和 Nikolaj Bjørner. 2008. Z3：一种高效的SMT求解器。载于《国际系统构造与分析工具及算法会议》（*TACAS*）。柏林海德堡，337–340。https://doi.org/10.1007/978-3-540-78800-3_24

C. R. Ramakrishnan 和 Jakob Rehof（编）。Springer，337–340。 https://doi.org/10.1007/978-3-540-78800-3_24

Jean-Yves Girard。1987。线性逻辑。《理论计算机科学》 50（1987），1–102。 https://doi.org/10.1016/0304-3975(87)90045-4

K Rustan M Leino. 2010. Dafny：一种用于验证函数正确性的自动化程序验证器。载于 国际逻辑编程、人工智能与推理会议（*LPAR*），塞内加尔达喀尔。Springer，348–370。S

从算法博弈论到分布式系统：具有机械化复杂性保证。在欧洲编程研讨会（*ESOP*）- amuel Merten、Alexander Baghaii 和 Gordon Stewart（计算机科学讲义，第 *10801* 卷），Amal Ahmed（编）。Springer-Verlag，561–588。https://doi.org/10.1007/978-3-319-89884-1_20

Jack Robertson 和 William Webb. 1998. 蛋糕切割算法：能公平分配时就要公平. AK Peters/CRC Press.

Erel Segal-Halevi、Avinoam Hassidim 和 Yonatan Aumann. 2018. 无界蛋糕上的公平分配：有限块中的简单算法和有界时间算法。*ACM*算法汇刊 13, 1 (2016),12:1–12:32. https://doi.org/10.1145/2988232

尼基尔·斯瓦米纳坦、Yuyan Wang 和 Justin Hsu. 2023. 用于概率编程的精确且可组合的采样器。载于 *ACM 程序设计语言会议，OOPSLA*。https://doi.org/10.1145/2499370.2491978

EminaTorlak 和 Rastislav Bodík。2013 年。使用 rosette 扩展求解器辅助语言。载于 *ACM* 关于编程与软件新思想、新范式及反思的国际研讨会（*Onward!*），印第安纳州印第安纳波利斯，Antony L. Hosking、PatrickTh. Eugster 和 Robert Hirschfeld（编）。ACM，135–152。 https://doi.org/10.1145/2509578.2509586G

埃哈德 J. 沃辛格和吉瓦尔日·斯加尔. 2007. 关于切蛋糕的复杂性. 离散优化 4, 2 (2007),213–220. https://doi.org/10.1016/j.disopt.2006.07.003

A 被省略的证明

这里我们详细说明第 4 节中的证明。为了逐步展开主要证明，我们需要几个技术引理。首先，约束转换只会在 y_{k+1} 和 y_{k+N}（e） 之间创建逻辑变量。

引理 A.1。 令 e 为任意表达式。对于任意 $k \in \mathbb{N}$，我们有 $G_Q(k, e) = n(e) + k$ 并且

$$\mathsf{FV}(c(k, e, \mathrm{ret})) \cap \mathcal{Y} = \mathsf{FV}(G_s(k, e)) \cap \mathcal{Y} = \mathsf{FV}(G_\rho(k, e)) \cap \mathcal{Y} = \{y_{k+1}, \ldots, y_{k+n(e)}\}.$$

因此， $N(e) = |\mathsf{FV}(c(k, e, \mathrm{ret})) \cap \mathrm{Y}|$ 对于任意 k。其次， e 的自由变量恰好是 X 中出现在 e 的约束中的逻辑变量。

引理 A.2. $\mathsf{FV}(e) = \mathsf{FV}(c(k, e, \mathrm{ret})) \cap \mathrm{X}$ 对于任意 $k \in \mathbb{N}$.

这两个引理均可通过对 e 进行归纳法得出。接下来，如果替换 S 的定义域涵盖 e 中的所有自由变量，那么 S 会替换掉 e 中约束里的所有程序变量。请注意，这并非显而易见——替换会用逻辑项替换变量，而逻辑项可能包含其他变量——但替换的结构确保了所有程序变量都会被替换。

引理 A.3。 令 S 为一个替换，使得 $\mathsf{FV}(e) \subseteq dom(S)$。则

$$\mathsf{FV}(c(k, e, \mathrm{ret})S) \cap \mathcal{X} = \emptyset.$$

证明。令 $S = \{t_n/x_n\} \cdots \{t_1/x_1\}$。我们证明一个更一般的事实：若 φ 是一个公式，且 $\mathsf{FV}(\varphi) \cap \mathbb{X} \subseteq \mathrm{dom}(S)$，则

$$\mathsf{FV}(\varphi\{t_n/\widehat{x_n}\} \cdots \{t_{n-i+1}/\widehat{x_{n-i+1}}\}) \cap \mathcal{X} \subseteq \{x_1, \ldots x_{n-i}\} \tag{3}$$

对于 $i = 0, \ldots, n$ 对 i 使用归纳法。基例根据定义可得：

$$\mathsf{FV}(\varphi) \cap \mathcal{X} \subseteq \mathrm{dom}(S) = \{x_1, \ldots, x_n\}.$$

假设该命题对 $i-1$ 成立，那么我们有：

$$\begin{aligned}
&\mathsf{FV}(\varphi\{t_n/\widehat{x_n}\} \cdots \{t_{n-(i-1)+1}/\widehat{x_{n-(i-1)+1}}\}\{t_{n-i+1}/\widehat{x_{n-i+1}}\}) \cap \mathcal{X} \\
&= \mathsf{FV}(\varphi\{t_n/\widehat{x_n}\} \cdots \{t_{n-(i-1)+1}/\widehat{x_{n-(i-1)+1}}\}) \setminus \{x_{n-i+1}\} \cup \mathsf{FV}(\{t_{n-i+1}\}) \cap \mathcal{X} \\
&= (\mathsf{FV}(\varphi\{t_n/\widehat{x_n}\} \cdots \{t_{n-(i-1)+1}/\widehat{x_{n-(i-1)+1}}\}) \cap \mathcal{X}) \setminus \{x_{n-i+1}\} \cup (\mathsf{FV}(\{t_{n-i+1}\}) \cap \mathcal{X}) \\
&\subseteq \{x_1, \ldots, x_{n-i+1}\} \setminus \{x_{n-i+1}\} \cup \{x_1, \ldots, x_{n-i}\} \\
&= \{x_1, \ldots x_{n-i}\}
\end{aligned}$$

如所愿。因此，令 i 为 n，我们得出结论：

$$\mathsf{FV}(\varphi\{t_n/\widehat{x_n}\} \cdots \{t_1/\widehat{x_1}\}) \cap \mathcal{X} \subseteq \emptyset.$$

应用引理 A.2，我们的断言得证。 □

定理 *4.5*（可靠性）。令 e 为一个类型正确的表达式，σ 为满足 $FV(e) \subseteq dom(\sigma)$ 的环境，S 为一个替换，V 为一个估值函数集合，$v \in$ 为 V，以及 μ 为一个来自 M_V、将 σ 与 S 结合起来的解释。如果 $\langle e, \sigma\rangle \Downarrow_V v$，那么对于任意 $k \geq m(S)$，存在一个来自 M_V 的解释 μ'，它在 $\{y_1, \ldots, y_k\}$ 上与 μ 一致，使得 $\mu' \vDash c(k, e, \widehat{v})S$。

证明。我们对 $e, \sigma \Downarrow_V v$ 的推导进行归纳法。假设 σ 和 S 都在 $x_1, \ldots, x_n$ 上。我们根据 $e, \sigma \Downarrow_V v$ 的求值规则进行分类讨论。

[瓦尔] 对于任意 k，我们都有 $c(k, e, \widehat{v}) = c(k, v, \widehat{v}) = (\widehat{v} = \widehat{v})$，因此 $\mu \vDash c(k, e, \widehat{v})$。

[变量] 我们有 $v = \sigma(x)$。因为 μ 与 σ 和 S 结婚，$\mu(xS) = \sigma(x)$，因此

$$\mu \vDash (\widehat{\sigma(x)} = \widehat{x})S.$$

注意到 $c(k, x, \sigma(x)) =$ 为真 $\wedge (\sigma(x) = \widehat{x})$ 完成了这一情形，并令 $\mu' = \mu$。

[Let] 我们有 $e =$（令 $x = e_1$ 于 e_2 中）使得 $e_1, \sigma \Downarrow_V v_1$ 以及 $e_2, \sigma' \Downarrow_V v$，其中我们设定 $\sigma' = \sigma[x \mapsto v_1]$。由于 $\mathsf{FV}(e_1) \subseteq \mathsf{FV}(e) \subseteq \mathrm{dom}(\sigma)$，且 μ 将 σ 与 S 结合，我们可以应用归纳假设，得到一个解释 $\mu_1 \in \mathrm{M}_V$ 使得

$$\mu_1 \vDash c(k, e_1, \widehat{v_1})S.$$

展开后，我们得到

$$\mu_1 \vDash (\mathrm{G}_\rho(k, e_1) = \widehat{v_1})S,$$

可以表示为

$$\mu_1(\widehat{x}S') = \sigma'(x),$$

其中我们令 $S' = \{\mathrm{G}_\rho(k, e_1)/\widehat{x}\}S$。这表明 μ_1 也使 σ' 和 S' 成立。还要注意，$\mathsf{FV}(e_2) \subseteq \mathrm{dom}(\sigma')$。因此，根据归纳法，对于 $k' = k + N(e_1) = m(S')$（由引理 A.1 得出，且因为 $k \geq m(S)$），存在某个来自 M_V 的适当解释 μ'，使得

$$\mu' \vDash c(k', e_2, \widehat{v})S', \tag{4}$$

且 μ' 与 μ_1 至多相差 k'。根据引理 A.1，$\mathsf{FV}(c(k, e_1, \widehat{v_1})S) \subseteq \{y_1, \ldots\ldots, y_{k'}\}$，因此还可得到

$$\mu' \vDash c(k, e_1, \widehat{v_1})S. \tag{5}$$

结合（5）与（4），我们得到

$$\mu' \vDash \mathrm{G}_s(k, e_1)S \wedge \mathrm{G}_s(k', e_2)S' \wedge (\widehat{v} = \mathrm{G}_\rho(k', e_2))S'.$$

回顾我们对 k'和 S'的定义，可以注意到上式恰好是

$$\mu' \vDash c(k, e, \widehat{v})S.$$

最后，由于 μ'与 μ_1在 k'之前一致，且 μ_1与 μ在 k之前一致，我们得出 μ'与 μ在 k之前一致，因此该情形得证。

[运算] 我们有 $v = [\![o]\!](v_1, \ldots\ldots, v_n)$、$e = o(e_1, \ldots\ldots, e_n)$，以及 e_i，$\sigma \Downarrow_V v_i$，其中 $i = 1, \ldots\ldots, n$，对于某些 $v_1, \ldots\ldots, v_n \in \mathrm{V}$、$o \in \mathrm{O}$ 和 $e_1, \ldots\ldots, e_n$。根据归纳法，我们有一个 $\mu_i \in \mathrm{M}_V$，使得

$$\mu_i \vDash c(k + N(e_1, \ldots, e_{i-1}), e_i, \widehat{v_i})S \tag{6}$$

且 μ_i与 μ在 k之前一致，对所有 $i = 1, \ldots\ldots, n$。根据引理 A.1，所有约束中的 $\mathcal{Y}$ 变量均成对不相交。因此，我们可以从 μ' 中 M_V 得到一个解释，使其与 μ 一致直到 k，并且

$$\mu' \vDash c(k + N(e_1, \ldots, e_{i-1}), e_i, \widehat{v_i})S$$

对于 $i = 1, \ldots, n$。回顾 e 的约束定义，我们有 $\mu' \vDash \mathrm{G}_s(k + N(e_1, \ldots, e_{i-1})\ e_i)S$ 以及 $\mu'(\mathrm{G}_\rho(k + N(e_1, \ldots, e_{i-1})\ e_i)S) = v_i$ 对于 $i = 1, \ldots, n$。因为 $\mu'(o) = [\![o]\!]$，由于 μ' 是适当的，我们也有

$$\mu'(o(\mathrm{G}_\rho(k, e_1), \ldots, \mathrm{G}_\rho(k + N(e_1, \ldots, e_{n-1}), e_n))S) = [\![o]\!](v_1, \ldots, v_n) = v,$$

因此我们可以得出结论：

$$\mu' \vDash c(k, e, \widehat{v})S.$$

这些情况的 [配对], [Proj], [蛋糕], [Lt], [Rt],以及 [Eval]都以几乎相同的方式处理，和[Ops] 一样，因此我们省略它们。

[IfTrue] 我们有 $e =$（若 e_1 则 e_2，否则 e_3），并且 $e_1, \sigma \Downarrow_V$ 为真，且 $v = v_2$。根据归纳法，存在 $\mu_1 \in \mathrm{M}_V$ 使得

$$\mu_1 \vDash c(k, e_1, \widehat{\mathsf{true}})S$$

以及 $\mu_2 \in \mathrm{M}_V$ 使得

$$\mu_2 \vDash c(k + N(e_1), e_2, \widehat{v_2})S$$

且二者都与 μ 一致，直到 k。我们可以通过设定来构造一个解释 $\mu' \in \mathrm{M}_V$：$\mu'(y_j) = \mu_1(y_j)$ 对于 $j = 1, \ldots, k + N(e_1)$，以及 $\mu'(y_j) = \mu_2(y_j)$ 对于所有其他的 j。利用引理 A.1，我们有

$$\mu' \vDash c(k, e_1, \widehat{\mathsf{true}})S \wedge c(k, e_2, \widehat{v_2})S. \tag{7}$$

这尤其意味着 $\mu'(\mathrm{G}_\rho(k,\ e_1)S) =$ 为真，并且 $\mu'(\mathrm{G}_\rho(k + N(e_1)\ e_2)S) = v_2$。根据 ite 的解释，我们有

$$\mu'(\mathrm{ite}(\mathrm{G}_\rho(k, e_1), \mathrm{G}_\rho(k + N(e_1), e_2), \mathrm{G}_\rho(k + N(e_1, e_2), e_3))S) = \widehat{v_2}. \tag{8}$$

再次使用（7）我们还有

$$\mu' \vDash \mathrm{G}_s(k, e_1)S \wedge \mathrm{G}_s(k + N(e_1), e_2)S \wedge (\mathrm{G}_\rho(k, e_1)S = \mathsf{true}),$$

这意味着

$$\mu' \vDash (\mathrm{G}_s(k, e_1)S \wedge \mathrm{G}_s(k + N(e_1), e_2)S \wedge (\mathrm{G}_\rho(k, e_1)S = \mathsf{true})) \\ \vee (\mathrm{G}_s(k, e_1)S \wedge \mathrm{G}_s(k', e_3)S \wedge (\mathrm{G}_\rho(k, e_1)S = \mathsf{false}))$$

但现在我们可以提取出 $\mathsf{G}_s(k,\ e_1)$ 和 S，从而得到

$$\begin{aligned}\mu' \vDash (&\mathrm{G}_s(k, e_1)\\ &\wedge (\mathrm{G}_s(k + N(e_1), e_2) \wedge (\mathrm{G}_\rho(k, e_1) = \mathsf{true}))\\ &\vee (\mathrm{G}_s(k + N(e_1, e_2), e_3) \wedge (\mathrm{G}_\rho(k, e_1) = \mathsf{false})))S.\end{aligned}$$

将此与（8）结合起来，我们得到

$$\mu' \vDash c(k, e, \widehat{v})S$$

如愿以偿。

[IfFalse] 与 [IfTrue]类似。

[标记] 我们有 $e =$标记$_a(e_1, e_2)$ 和 $e_1, \sigma \Downarrow_V v_1$ 以及 $e_2, \sigma \Downarrow_V v_2$，还有 $e, \sigma \Downarrow_V r$，其中某个 $r \in \mathbb{R}$ 满足条件。此外，我们有 $V_a([v_1, 1]) \geq v_2$ 以及 $V_a([v_1, r]) = v_2$。利用归纳假设，存在 μ_1 和 μ_2，它们来自 $\mathbb{M}_V$，使得 $\mu_1 \vDash c(k, e_1, \widehat{v_1})S$ 且 $\mu_2 \vDash c(k + N(e_1)\ e_2, \widehat{v_2})S$，并且二者都与 μ 在 k 以内一致。从这里我们可以由引理 A.1 看出，$\mathsf{FV}(c(k+N(e_1)\ e_2, \widehat{v_2})) \cap \mathsf{FV}(c(k, e_1, \widehat{v_1})) \cap \mathcal{Y} = \emptyset$。因此，利用 μ_1 和 μ_2，我们可以构造出一个来自 $\mathbb{M}_V$ 的 μ'，它与 μ 在 k 以内一致；特别地，$\mu'(y_{k'}) = r$，其中 $k' = k + N(e_1) + 1$，并且

$$\mu' \vDash \mathrm{G}_s(k, e_1')S \wedge \mathrm{G}_s(k + N(e_1'), e_2')S \wedge \mathrm{Val}_a([v_1, 1] \geq \widehat{v_2}) \wedge (\mathrm{Val}_a([v_1, y_{k'}]) = \widehat{v_2}) \wedge (\widehat{v} = y_{k'}).$$

提取出 S，我们得到 $\mu' \vDash c(k, e, \widehat{v})S$，正如所需。

[除法规则] 我们有 $v = ([r_1, r_2],\ [r_2, r_1])$、$r_1 \leq r_2 \leq r_1$，$e =$除法$(e_1,\ e_2)$，以及两个$e_1$,$-\ \sigma \Downarrow_V [r_1, r_1]$和$e_2,\ \sigma \Downarrow_V r_2$。由归纳法，我们有 $\mu_1,\ \mu_2 \in \mathbb{M}_V$，使得

$$\mu_1 \vDash c(k, e_1, \widehat{[r_1, r_1']})S \quad \text{and} \quad \mu_2 \vDash c(k + N(e_1), e_2, \widehat{r_2})S \tag{9}$$

且二者都 μ_1和 μ_2均与 μ在 $\{y_1$、……、$y_k\}$。根据引理 A.1，

$$\mathsf{FV}(c(k, e_1, \widehat{[r_1, r_1']})) \cap \mathsf{FV}(c(k + N(e_1), e_2, \widehat{r_2})) \cap \mathcal{Y} = \emptyset.$$

因此，我们可以从 $\mathbb{M}$ 获得一个解释 μ'_V，它在 $\{y_1$、……、$y_k\}$ 上与 μ 一致，并且满足以下条件。

$$\mu' \vDash c(k, e_1, \widehat{[r_1, r_1']})S \wedge c(k + N(e_1), e_2, \widehat{r_2})S$$

回顾 e的约束定义，我们有 $\mu' \vDash \mathrm{G}_s(k, e_1)S \wedge \mathrm{G}_s(k + N(e_1)\ e_2)S$以及$\mu'(\mathrm{G}_\rho(k, e_1)S) = [r_1, r_1']$以及 $\mu'(\mathrm{G}_\rho(k + N(e_1)\ e_2)S) = r_2$。由于 $\mu'(\ell)$ 和 $\mu'(r)$ 分别给出了区间的左端点和右端点，我们还可得 $\mu'(\ell(\mathrm{G}_\rho(k, e_1)S)) = r_1$以及 $\mu'(r(\mathrm{G}_\rho(k, e_1)S)) = r_1'$。因此，由于 $r_1' \geq r_2 \geq r_1$，并再次利用 μ' 是适当的，不难看出

$$\mu' \vDash c(k, e, \widehat{v})S.$$

[配对], [投影], [蛋糕], [左], [右],的情况以及 [求值]的情况都与[运算] 几乎完全相同，因此省略。□

定理 4.7（完备性）。设 e 是一个类型正确的表达式，S 是一个替换，满足 $\mathsf{FV}(e) \subseteq dom(S)$，$V$ 是一组估值函数，$v \in \mathcal{V}$，且 σ 是一个环境。若存在 $k \geq 0$ 和一个从$\mathbb{M}_V$到 σ与 S的解释，使得 $\mu \vDash c(k, e, \widehat{v})S$，则$\langle e, \sigma\rangle \Downarrow_V v$。

证明。我们按 e的语法进行归纳。假设 $\mu \vDash c(k, e, \widehat{v})S$，其中 S是定义在 $x_1, \ldots, x_n$上的替换，而 σ是定义在 $x_1, \ldots, x_n$ 上的环境，且 μ将 σ与 S配对。我们分情况讨论。

情况 $e = v$**：** 立即。

情况 $e = x$**：** 我们有 $x \in \mathrm{dom}(S) = \mathrm{dom}(\sigma)$。由于 $c(k,\ e,\ \widehat{v}) = \mathrm{true} \wedge (\widehat{x} = \widehat{v})$，我们有 $\mu(\widehat{x}S) = v$。由于 μ与 σ和 S结婚，我们有 $\sigma(x) = \mu(\widehat{x}S)$。由于$x$，$\sigma$根据 [Var],$\Downarrow_V \sigma(x)$，我们就完成了这个情况。

情形 $e = (\text{令 } x = e_1 \text{ 属于 } e_2)$**：** 我们有

$$\mu \vDash \mathrm{G}_s(k, e_1)S \wedge \mathrm{G}_s(k + N(e_1), e_2)S' \wedge (\widehat{v} = \mathrm{G}_\rho(k + N(e_1), e_2))S',$$

其中我们令 $S' = \{\mathrm{G}_\rho(k, e_1)/x\}S$。因此，$\mu \vDash c(k, e_1, \widehat{v_1})S$和 $\mu \vDash c(k + N(e_1)\ \widehat{v_2})S'$均成立，其中我们令 $v_1 = \mu(\mathrm{G}_\rho(k, e_1)S)$，以及 $v_2 = \mu(\mathrm{G}_\rho(k + N(e_1)\ e_2)S')$。于是，$\mu$将 σ' 与 S' 配对，其中我们令 $\sigma' = \sigma[x \mapsto v_1]$。因此，根据两次归纳法假设，我们有 $e_1, \sigma \Downarrow_V v_1$ 以及 $e_2, \sigma' \Downarrow_V v_2$。但随后我们可以通过 [[]Let] 得到 $e, \sigma \Downarrow_V v_2$。由于 $\mu(\widehat{v}) = \mu(\widehat{v_2})$，我们有 $v = v_2$，因此这一情形得证。

情形 $e = o(e_1, \ldots, e_n)$：我们有 $\mu \vDash c(k + N(e_1, \ldots, e_{i-1})\ e_i, \widehat{v_i})S$，对于 $i = 1, \ldots, n$，其中 $v_i = \mu(\mathrm{G}_\rho(k + N(e_1, \ldots, e_{i-1})\ e_i)S)$。根据归纳法，我们有 $e_i, \sigma \Downarrow_V v_i$对于 $i = 1, \ldots, n$，这意味着 $e, \sigma \Downarrow_V [\![o]\!](v_1, \ldots, v_n)$，利用了 $\overline{e}$是类型正确的，并且 [Ops]。由于 $\mu \vDash c(k, e, \widehat{v})S$，我们有

$$v = \mu(\widehat{v}) = \mu(o(\mathrm{G}_\rho(k, e_1), \ldots, \mathrm{G}_\rho(k + N(e_1, \ldots, e_{n-1}), e_n))S) = [\![o]\!](v_1, \ldots, v_n),$$

所以我们已经处理完这种情况。

情形 $e = (\text{如果 } e_1 \text{ 则 } e_2 \text{ 否则 } e_3)$：我们有 $\mu \vDash c(k, e_1, \widehat{v_1})S$，其中 $v_1 = \mu(\mathrm{G}_\rho(k, e_1)S)$。由于 e类型正确，我们要么有 $v_1 = \text{真}$，要么有 $v_1 = \text{假}$。假设是前一种情况。根据归纳法，我们有$e_1, \sigma \Downarrow_V$真，并且我们有 $\mu \vDash c(k + N(\overline{e_1})\ e_2, \widehat{v})S$，使用 ite 的定义以及$\mu \vDash c(k, e, \widehat{v})$。根据归纳法，$e_2, \sigma \Downarrow_V v$。因此，$e, \sigma \Downarrow_V v$，利用 e类型正确这一点，从而我们可以应用 [IfTrue]。带有 $v_1 = \text{假}$的另一种情况类似。

情形 $e = \text{标记}_a(e_1, e_2)$：我们同时有 $\mu \vDash c(k, e_1, \widehat{v_1})S$和 $\mu \vDash c(k + N(e_1)\ e_2, \widehat{v_2})S$，其中我们设定$v_1 = \mu(\mathrm{G}_\rho(k, e_1)S)$，并且 $v_2 = \mu(\mathrm{G}_\rho(k + N(e_1)\ e_2)S)$。根据归纳法，我们有 $e_1, \sigma \Downarrow_V v_1$和 $e_2, \sigma \Downarrow_V v_2$。现在令 $r = \mu(y_{k+N(e_1,e_2)+1})$。由于 $\mu \vDash c(k, e, v)S$，我们也有 $V_a([v_1, r]) = v_2$。但这意味着，根据 [标记]，$e, \sigma \Downarrow_V r$。

情形 $e = \text{除法}(e_1, e_2)$：我们有 $\mu \vDash c(k, e_1, [r_1, r_1'])S$，以及 $\mu \vDash c(k + N(e_1)\ e_2, \widehat{r_2})S$，其中我们令$r_1 = \mu(\ell(\mathrm{G}_\rho(k, e_1)S))$、$r_1' = \mu(r(\mathrm{G}_\rho(k, e_1)S))$，以及 $r_2 = \mu(\mathrm{G}_\rho(k + N(e_1)\ e_2)S)$。根据归纳法且由于 μ是适当的，我们有 $e_1, \sigma \Downarrow_V [r_1, r_1']$以及 $e_2, \sigma \Downarrow_V r_2$。利用 $\mu \vDash \mathrm{G}_s(k, e)S$，我们有 $r_1' \geq r_2 \geq r_1$。因此，我们可以应用 [除法规则] 来得到$e, \sigma \Downarrow_V ([r_1, r_2], [r_2, r_1'])$。利用 μ'是适当的，我们可以仿照 $e = o(e_1, \ldots, e_n)$的情形进行论证，从而得到$v = ([r_1, r_2], [r_2, r_1'])$，因此这一情形得证。

其余情况类似。 □

B 实现细节

约束转换：条件表达式。我们的实现采用了一种略有不同的条件表达式转换方式，如图所示 12。不难看出，这些约束在逻辑上等价于图 4中的条件表达式转换方式，但我们发现，使用此处的转换方式时，Z3 的表现要好得多。

$$
\begin{aligned}
\mathrm{G}_Q(k, \text{if } e_1 \text{ then } e_2 \text{ else } e_3) &\triangleq k + N(e_1, e_2, e_3) + 1 \\
\mathrm{G}_s(k, \text{if } e_1 \text{ then } e_2 \text{ else } e_3) &\triangleq (((\mathrm{G}_\rho(k, e_1) = \text{true}) \wedge \mathrm{G}_s(k + N(e_1), e_2) \\
&\quad \wedge (y_{k+N(e_1,e_2,e_3)+1} = \mathrm{G}_\rho(k + N(e_1), e_2)) \\
&\quad \vee ((\mathrm{G}_\rho(k, e_1) = \text{false}) \wedge \mathrm{G}_s(k + N(e_1, e_2), e_3) \\
&\quad \wedge (y_{k+N(e_1,e_2,e_3)+1} = \mathrm{G}_\rho(k + N(e_1, e_2), e_3))) \\
&\quad \wedge \mathrm{G}_s(k, e_1) \\
\mathrm{G}_\rho(k, \text{if } e_1 \text{ then } e_2 \text{ else } e_3) &\triangleq y_{k+N(e_1,e_2,e_3)}
\end{aligned}
$$

图 12. 约束转换。

关于估值的公理。 我们的实现使用了以下估值公理:

(1) $V([0,1]) = 1$(2) $V(I) \geq 0$(3) $(\ell(I_2) = r(I_1))$ $\Rightarrow V(I_1) + V(I_2) = V([\ell(I_1)\ r(I_2)])$ (4) $I_2 \subseteq I_1 \Rightarrow V(I_2) \leq V(I_1)$ (5) $V(I) \leq 1$(6) $(I = \{r\}) \Rightarrow V(I) = 0$ (7) $\ell(I_1) \geq \ell(I_2)\ V(I_1) \geq V(I_2) \Rightarrow r(I_1) \geq r(I_2)$ (8) $r(I_1) \geq r(I_2)\ V(I_2) \geq V(I_1) \Rightarrow \ell(I_1) \geq \ell(I_2)$

对于任意区间 I， I_1， $I_2 \subseteq [0,1]$。前三条公理对应第2节中的前三个假设。连续性假设已内置于为标记生成的约束中；事实上，Robertson-Webb 模型中的所有协议都要求对估值作出这一假设。其余假设是关于估值的一般事实；虽然可以从前三条公理推导出这些假设，但我们发现，将它们纳入对 Z3 很有用。

C 用于验证进展的自定义约束转换

我们在第4 节中看到了如何验证与协议输出相关的性质。那么，与输出无关的协议性质又该如何验证呢？进展就是这样一种性质，也就是说，只要协议类型正确，它就总能执行一步并得到一个值。协议可能是无嫉妒的，即每当它执行一步得到一个分配时，该分配都是无嫉妒的，但协议仍可能陷入停滞。这显然不是理想的行为，因此验证进展很有价值。

第 4 节中的完备性可用于证明这一点，然而这将需要证明 $c(e、\text{ret})$ 在 $\underline{\mathrm{M}}_V$中对于每组估值函数 V都是可满足的。这种方法不太容易适用于 SMT求解器。在本节中，我们提出另一种更容易适用于 SMT 求解器的翻译。

更明确地说，进展是指协议总会向某个值迈进的性质。这意味着，每当 $e = \text{标记}_a(e_1, e_2)$ 且 $e_1, \sigma \Downarrow v_1$、$e_2, \sigma \Downarrow v_2$时，我们要求存在 $r \in [0,1]$使得 $V_a([v_1, r]) = v_2$；并且，每当 $e = \text{除法}(e_1, e_2)$，且 $e_1, \sigma \Downarrow [r_1, r_1'], e_2, \sigma \Downarrow r_2$ 时，必须有 $r_1 \leq r_2 \leq r'1$。鉴于定理 4.5，如果我们能够证明 $c(k,(e_1, e_2), (v_1, v_2))$ 蕴含 $\mathrm{Val}_a([v_1, 1]) \geq v_2$，并且 $c(k,(e_1, e_2), (r_1, r_2))$ 蕴含 $r_1' \geq r_2 \geq r_1$，那么我们就

$m_a(k, e_1, e_2, B, S) \triangleq \forall \text{ret}.\exists y_1, \ldots, y_{k+N(e_1,e_2)}.(c(k, (e_1, e_2), \text{ret})S \wedge B) \Rightarrow (V_a([\pi_1 \text{ret}, 1]) \geq \pi_2 \text{ret}))d(k, e_1, e_2, B, S) \triangleq \forall \text{ret}.\exists y_1, \ldots, y_{k+N(e_1,e_2)}.(c(k, (e_1, e_2), \text{ret})S \wedge B) \Rightarrow (\ell(\pi_1 \text{ret}) \leq \pi_2 \text{ret} \leq r(\pi_1 \text{ret})))$

$\alpha(k, \text{标记}_a(e_1, e_2)\ B, S) \triangleq \alpha(k, e_1, B, S) \wedge \alpha(k, e_2, B, S) \wedge m_a(k, e_1, e_2, B, S)$

$\alpha(k, \text{除法}(e_1, e_2)\ B, S) \triangleq \alpha(k, e_1, B, S) \wedge \alpha(k, e_2, B, S) \wedge d(k, e_1, e_2, B, S)$

$\alpha(k, \text{若 } e_1 \text{则 } e_2 \text{否则 } e_3, B, S) \triangleq \alpha(k, e_1, B, S)$

$\wedge\ \alpha(k + N(e_1), e_2, (\mathrm{G}_\rho(k, e_1) = \text{true})S \wedge \mathrm{G}_s(k, e_1)S \wedge B, S)$

$\wedge\ \alpha(k + N(e_1), e_3, (\mathrm{G}_\rho(k, e_1) = \text{false})S \wedge \mathrm{G}_s(k, e_1)S \wedge B, S)$

$\alpha(k, \text{令 } x = e_1 \text{为 } e_2, B, S) \triangleq \alpha(k, e_1, B, S) \wedge \alpha(k + N(e_1)\ e_2, B \wedge \mathrm{G}_s(k, e_1)S, \{\mathrm{G}^\rho(k, e_1)/\widehat{x}\}S)\alpha(k, v, B, S) \triangleq \text{true}\alpha(k, x, B, S) \triangleq \text{true}\alpha(e(e_1, \ldots, e_n)\ B, S) \triangleq \alpha(k, e_1, B, S) \wedge \cdots \wedge \alpha(k, e_n, B, S)$ 对所有其余表达式均成立。

图 13。约束转换：进展。

能够证明某个协议始终有进展。然而，分支条件也可能有助于证明这些事实。这些考虑促成了以下定义。

令 α： $\mathbb{N} \times \mathsf{Exp} \times \mathsf{Form} \times \mathsf{S} \to \mathsf{Form}$ 如图 13所示，定义在表达式上，其中 S 是替换的集合。我们给出一个高层次的解释。我们希望，如果 $\alpha(k, e, B, S)$ 被解释为真，那么 e 总是会步进到一个值。这就是为什么除了标记、除法规则和 if then else 以外的所有表达式，α 都只会传递到其子表达式。B 可以理解为一个公式，用于跟踪协议中的分支条件和先前的副条件，因此 B 只有在条件表达式和 let 绑定向子表达式传递时才会更新。替换 S 是正确跟踪绑定的程序变量所必需的，这与第 4节中的用法类似。我们希望只保留足够的事实，以便始终能够证明进展。下面的例子或许会有所启发。

例。 再来看图3中的切分—选择。我们有

$\alpha(0, e, \text{true}) = \forall \text{ret}。(\text{ret} = (0, 1/2)) \Rightarrow (\text{Val}_1([\pi_1 \text{ret}, 1]) \geq \pi_2 \text{ret}) \wedge \forall \text{ret}.(\exists y_1.(\text{Val}_1([0, y_1]) = 1/2) \wedge (\text{Val}_1([0, 1]) \geq 1/2) \wedge (\text{ret} = y_1)) \Rightarrow r(\pi_1(\text{ret})) \geq \pi_2(\text{ret}) \geq \ell(\pi_1(\text{ret}))).$

这里我们省略了所有与 true 合取的项。第一行对应标记子表达式，第二行对应除法子表达式。显然，任何适当的解释都能满足$\alpha(0, e, \text{true})$。

现在我们通过一些有用的技术事实，逐步证明 α 实现其目的。

引理 C.1。 对于任意公式 B和替换 S，满足$\mathsf{FV}(B) \cup \mathsf{FV}(S) \subseteq \{y_1, \ldots, y_k\}$，

$$\mathsf{FV}(\alpha(k', e, B, S)) \cap \mathcal{Y} = \emptyset$$

如果 $k' \geq k$。

证明。对 e的结构进行归纳法。 □

引理 C.2. 设 e 为任意表达式， B 为任意公式， S 为任意替换，且$\mathsf{FV}(e) \subseteq dom(S)$，并且 $\mathsf{FV}(B)\mathcal{X} = \emptyset$。则$\mathsf{FV}(\alpha(k, e, B, S)) \cap \mathcal{X} = \emptyset$。

证明。对 e的结构进行归纳法。 □

鉴于这些事实，我们现在陈述并证明所需的结果。

命题 C.3。 令 $\langle e, \sigma\rangle$ 为一个配置，使得 $\mathsf{FV}(e) \subseteq dom(\sigma)$。如果存在一个公式 B、替换 S 和解释 $\mu \in \mathrm{M}V$ 以及自然数 k 使得 ¯

(1) $\mathsf{FV}(B) \subseteq \{y_1, \ldots, y_k\}$

(2) $k \geq m(S)$*(3)* μ 与 σ 结

婚，和 S*(4)* $\mu \vDash \alpha(k, e, B, S)$

$\wedge B$

然后 $\langle e, \sigma\rangle \Downarrow_V \underline{v}$ 对于某个 $v \in \mathcal{V}$。

证明。我们根据子表达式关系进行归纳法证明。
如果 $e = v$ 或 $e = x$，我们就立即得证。

假设 e =标记$_a(e_1, e_2)$。根据 α的定义，我们有 $\mu \vDash \alpha(k, e_1, B, S)$和$\mu \vDash \alpha(k, e_2, B, S)$。根据归纳法，存在 v_1, v_2 使得 $e_1, \sigma \Downarrow_V v_1$，且 $e_2, \sigma \Downarrow_V v_2$。于是我们有 $(e_1, e_2)\ \sigma \Downarrow_V (\overline{v_1}, v_2)$。根据定理 4.5，我们有

$$\mu' \vDash c(k, (e_1, e_2), (v_1, v_2))S$$

对于某个与 μ 在 k之前一致的适当解释 μ'。因此我们可以说

$$\mu \vDash \exists y_1, \ldots, y_{k+N(e_1,e_2)}.c(k, (e_1, e_2), (v_1, v_2))S)$$

但由于 $\mu \vDash B$，我们也有 $\mu' \vDash B$，这是利用假设 (1) 以及 μ' 与 μ 在 k之前一致这一点得到的，因此

$$\mu' \vDash (c(k, (e_1, e_2), (v_1, v_2))S \wedge B.$$

根据引理A.1 以及假设 (2) 和上述内容，我们有

$$\mathsf{FV}(c(k, (e_1, e_2), (v_1, v_2))S \wedge B) \subseteq \{y_1, \ldots, y_{k+N(e_1,e_2)}\},$$

因此我们也得到

$$\mu \vDash \exists y_1, \ldots, y_{k+N(e_1,e_2)}.c(k, (e_1, e_2), (v_1, v_2))S \wedge B.$$

因此，由于 $\mu \vDash m_a(k, e_1, e_2, B, S)$，取 ret 为$(v_1, v_2)$，我们得到 $\mu \vDash \mathrm{Val}_a([\widehat{v_1}, 1]) \geq \widehat{v_2}$。这意味着存在某个 $r \in \mathbb{R}$，使得 $V_a([v_1, r]) = v_2$。于是我们可以得出$e, \sigma \Downarrow_V r$。 ¯

现在，如果 $e =$除法$(e_1,\ e_2)$，也可以提出一个非常相似的论证。

假设 $e =$如果 e_1 那么 e_2 否则 e_3。那么我们有

$$\mu \vDash \alpha(k, e_1, B, S),$$

$$\mu \vDash \alpha(k + N(e_1), e_2, (\mathrm{G}_\rho(k, e_1) = \mathsf{true})S \wedge \mathrm{G}_s(k, e_1)S \wedge B, S)$$

and

$$\mu \vDash \alpha(k + N(e_1), e_3, (\mathrm{G}_\rho(k, e_1) = \mathsf{false})S \wedge \mathrm{G}_s(k, e_1)S \wedge B, S).$$

立即通过归纳法，$\langle e_1,\ \sigma\rangle \Downarrow_V$ 为真 或 $\langle e_1,\ \sigma\rangle \Downarrow_V$ 为假。假设前者为真。那么根据定理 4.5，我们有存在 μ' 使得 $\mu' \vDash c(k,\ e_1,$ 为真$)S$并且 μ' 与 μ一致，直到 k。这意味着 $\mu' \vDash B'$，其中

$$B' = (\mathrm{G}_\rho(k, e_1) = \mathsf{true})S \wedge \mathrm{G}_s(k, e_1)S \wedge B.$$

注意到 $\mathsf{FV}(B') \cap \mathcal{X} \subseteq \mathrm{dom}(\sigma)$ 由引理 A.2 并且 $\mathsf{FV}(B') \cap \mathcal{Y} \subseteq \{y_1, \ldots, y_{k+N(e_1)}\}$ 由引理 A.1 并根据关于 B 的假设。因为

$$\mathsf{FV}(\alpha(k + N(e_1), e_2, (\mathrm{G}_\rho(k, e_1) = \mathrm{true})S \wedge \mathrm{G}_s(k, e_1)S \wedge B, S)) = \emptyset,$$

由引理C.2和引理C.1，我们还可以得到

$$\mu' \vDash \alpha(k + N(e_1), e_2, (\mathrm{G}_\rho(k, e_1) = \mathrm{true})S \wedge \mathrm{G}_s(k, e_1)S \wedge B, S),$$

因为 μ 和 μ' 都是适当的。因此我们可以应用归纳法得到 $e_2, \sigma \Downarrow_V v_2$，其中某个 v_2。这意味着 $e, \sigma \Downarrow_V v_2$。后一种情况的论证类似。

假设 $e =$ (令 $x = e_1$ 为 e_2)。那么 $\mu \vDash \alpha(k, e_1, B, S)$ 且 $\mu \vDash \alpha(k + N(e_1)\ e_2, B', S')$，其中我们设 $B' = B \wedge \mathrm{G}_s(k, e_1)S$ 且 $S' = \{\mathrm{G}_\rho(k, e_1)/x\}S$。我们可以立即应用归纳法，得到对于某个 v_1，$e_1, \sigma \Downarrow_V v_1$。根据定理 4.5，存在一个恰当的解释 μ'，使得

$$\mu' \vDash c(k, e_1, \widehat{v_1})S,$$

且 μ' 与 μ 至多相差 k。展开 e_1 的约束 $\mu'(\mathrm{G}\rho(k, e_1)S) = v_1$ 可见 μ' 与 $\sigma[x \mapsto v_1]$ 结婚，且 S'。现在根据引理 C.1 和引理 C.2，

$$\mathsf{FV}(\alpha(k + N(e_1), e_2, B', S')) = \emptyset,$$

所以我们还有

$$\mu' \vDash \alpha(k + N(e_1), e_2, B', S').$$

回想起 μ' 在 k 范围内与 μ 一致，且 $\mathrm{FV}(B) \cap \mathrm{Y} \subseteq \{y_1, \ldots, y_k\}$，我们也可得出 $\mu' \vDash B'$。因此，我们可以再次应用归纳法，得到 $e_2, \sigma[x \mapsto v_1] \Downarrow_V v_2$，其中某个 $v_2 \in \mathrm{V}$。这意味着 $e, \sigma \Downarrow_V v_2$。□

上述命题包含许多条件。正如我们之前所见，当一个表达式没有自由变量时，情况会简单得多。

推论 C.4. 设 e 是一个不含自由变量的表达式。若存在 $\mu \in \mathrm{M}V$ 使得

$$\mu \vDash \alpha(0, e, \mathrm{true}, \varepsilon),$$

那么对于某个 $v \in \mathcal{V}$，有 $\langle e, \varepsilon \rangle \Downarrow_V v$。

我们可以如下应用这一结果。给定一个不含自由变量的表达式，$\alpha(0, e, \mathrm{true}\ \varepsilon)$ 根据引理 C.2 和引理 C.1 也不含自由变量。因此，$\alpha(0, e, \mathrm{true}\ \varepsilon)$ 在 $\mathrm{M}V$ 中可满足，当且仅当它在 $\mathrm{M}V$ 中有效。所以，如果我们能证明 $\vDash \alpha(0, e, \mathrm{true}\ \varepsilon)$，那么对于任意 V，我们就证明了对于某个 $v \in \mathrm{V}$，有 $e, \varepsilon \Downarrow_V v$。可以采用与第 6 节所述类似的方法，使用 SMT 求解器检查 $\vDash \alpha(0, e, \mathrm{true}\ \varepsilon)$。

D SELFRIDGE-CONWAY-剩余协议

Selfridge-Conway-剩余协议遵循 Selfridge-Conway 协议，但会提前停止，将剩余区间弃置，而不是在参与者之间进行分配。图14 展示了该协议的代码，图15展示了一种可能的分配方案。

收稿于 2022-11-10；接受于 2023-03-31

```
塞尔弗里奇-康威法剩余 : let (I1, I′1) =除
法(cake, mark1(0, 1/3)) in let (I2, I3) =
除法(I′1, mark1(0, 2/3)) in let (A, B, C) =
sort2(I1, I2, I3) in if eval2(A) =eval2(B)
then let (A, B, C) =sort3(A, B, C) in let (
B, C) =sort2(B, C) in alloc(C, B, A) else
let m = mark2(左 A, eval2(B)) in let
(Trim, Rest) =除法(A, m) in let (A, B, C)
=sort3(Trim B, C) in if A ≠ Trim t
hen if B = Trim then alloc(C, B, A, )
else alloc(B, C, A) else let (B, C) =
sort2(B, C) in alloc(C, B, A)
```

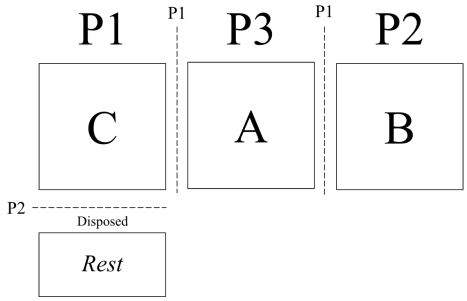


图 15。 来自
塞尔弗里奇-康威法剩余。

图 14. 塞尔弗里奇-康威法剩余 inSlice。