使用 Event-B 机械化拟运算

Christian Attiogbé

南特 大学、南特中央理工学院、法国国家科学研究中心、*LS2N*实验室、*UMR 6004*、南特、*F-44000*、法国

*2025*年*12*月

arXiv:2512.16342v1 [cs.LO] 18 Dec 2025

摘要

对自然系统和工业系统进行严谨建模仍面临诸多挑战，包括抽象、建模方法，以及用于构建、组合和推理模型的易用工具。算子是数学结构，可提供这类抽象，用于组合各种对象并确保其良构性。

算子的具体实现将提供实用手段，以便利用算子并将其应用于各种技术领域。我们从这些数学结构出发，使用 Event-B 开发了一条完整的精化链，实现了代数算子及其基本运算。

从方法论角度看，本工作的成果可用于处理符号计算问题中的类似实现，也可用于推理由算子结构支持的符号计算应用。

关键词： 算子，语义，符号计算，形式化建模，Event-B

1. 引言

对自然系统和工业系统进行严谨建模仍面临诸多挑战，包括抽象、建模方法，以及用于构建、组合和推理模型的易用工具。算子是数学结构，可提供这类抽象，用于组合各种对象并确保其良构性 [6]。

计算领域的一个关注点是如何构建和建模复杂的（基于组件的）系统及其演化。例如，考虑对一个制造系统进行建模，并通过将一个由人操作的模块替换为一个基于机器的模块来进一步扩展该系统。

动机。 我们的动机是将算子作为实用的系统建模工具，供（基于软件的）系统工程使用。事实上，形式化建模活动仍需要为从业者提供方法和通用库。 [3] 提出的挑战便体现了这一点。

目标。 本工作的目标是用 Event-B 实现通用结构和方法库，以支持形式化建模与分析。

贡献。 本文的贡献有多方面：*i)* 一个可实际操作算子的组件库；*ii)* 一个用于严谨构建对其他类似代数结构进行操作的框架。

文章结构。 本文组织如下。在第2.1节中，我们介绍算子和 Event-B 方法的背景。第3节专门介绍建模方法论。在第4.2节中，我们说明如何（重新）使用这项工作，最后第5节给出一些结论。

2. 背景

本节将介绍代数拟运算以及 Event-B。Event-B 是我们用于对算子进行建模和机械化的形式体系。

2.1. 算子

算子是一种抽象，用于编码代数结构，并考察这些结构上关系的复合。因此，算子能够操纵和推广代数结构。正如函数作用于集合一样，态射作用于算子；它们是对象的范畴。算子有很多种：拓扑算子群、代数算子、模算子群等 [6]。本文将重点讨论代数算子。

代数算子。一个基本示例是自同态算子群</style)，即 (End $_X, \circ_i$)，它定义在函数集合 End_X 上，其运算为 $\circ_i$，其中 X 是一个集合， i 是一个自然数。

算子 O_e =(*End* $_X, \circ_i$) 满足：
结束 $_X(k)$ = $X^k \to X$ 在 $k \geq 1$的条件下，是从 k 个 X 的乘积到 X的函数集合
$\circ_i$: 终止 $_X(n)$ ×终止 $_X(m)$ →终止 $_X(n+m-1)$ 是对 终止 $_X$上的函数进行复合的运算，其中有两个自然数 n和 m，且 $1 \leq i \leq n$。

给定 $f \in$终止 $_X(n)$ 和 $g \in$终止 $_X(m)$，运算 $\circ_i$定义如下

$$\begin{aligned}&\circ_i(f(x_1, x_2, \cdots, x_n), g(x_1, x_2, \cdots, x_m)))\\ &\quad = (f \circ_i g)(x_1, x_2, \cdots, x_{n+m-1})\\ &\quad = f(x_1, x_2, \cdots, x_{i-1}, g(x_i, x_{i+1}, \cdots, x_{i+m-1}), x_{i+m}, \cdots, x_{n+m-1})\end{aligned}$$

函数 $id_X : X \to X$ 是 o_i的单位元。对于 $f \in X^n \to X$，满足 $n \geq 1$、$f \circ_i id_X = f$ 和 $id_X \circ_1 f = f$。
终止 $_X(0)$ 的函数（或 $X^0 \to X$）对应于 X的元素；因此，它们是常量（设为 $c \in X$），给定 $f : X^n \to X$后可用于在 f中的指定位置 i 阻断复合；也就是说 $f \circ_i c = c$，其中 $i \leq n$。

此外，运算 $\circ_i$ 满足以下公理：顺序复合：$(f \circ_i g) \circ_{i-1+j} h = f \circ_i (g \circ_j h)$，当 $1 \leq i \leq n$ 且 $1 \leq j \leq m$ 时；并行复合：$(f \circ_i g) \circ_{k-1+m} h = (f \circ_k h) \circ_i g$，当 $1 \leq i < k \leq n$ 时

树歌剧（见图 1）和小立方体（见图 2）算子是表示（见 [9, 6]）的著名图形示例，能帮助人们直观理解算子操作。在这些示例中，请注意， $X = \{x_1, x_2, \cdots\}$ 的元素只写作 1、2、3 等，而 f、 g的图像或输出只用一个点表示。这是对X的一种抽象；（形式上，算子的默认唯一输出抽象为 0，而不写出）。

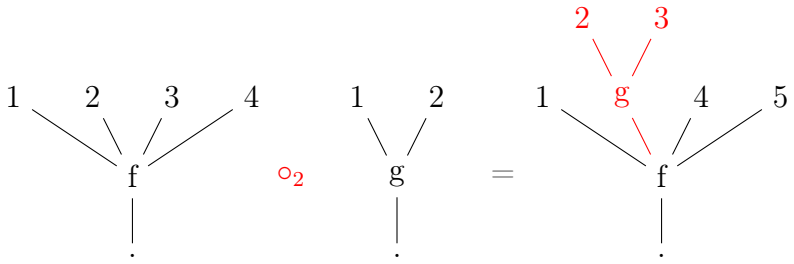

图 1：树状算子的复合

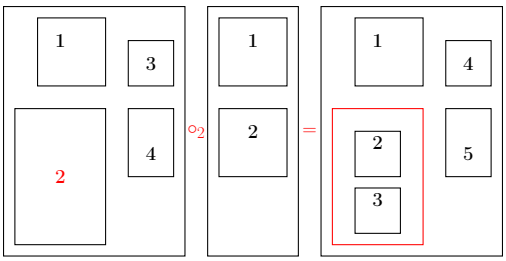

图 2：立方体算子的复合

为了说明复合算子 $\circ_i$，我们来看下面这个示例。制造系统的一种初步架构（见图 3）用一个含有 5 个参数的算子建模，这 5 个参数分别表示制造系统中由传送带连接的 5 个组件。参数 1 表示原材料来源；参数 2 表示一个模块，其中的机器会对从传送带上取出的物品施加特定的加工过程。参数 3 表示一个工作站，在此处人从传送带上取出物品，施加指定的加工过程，再将物品放回传送带。最后，最终产品到达作为参数 4 建模的仓库，而不合格产品则到达作为参数 5 建模的仓库。

接下来，我们希望在该制造系统的架构中，用一个由 2 台与一台机器相关联的机器人构成的新模块，替换由人工操作员操作的模块 3；第一台机器人从传送带上收集物品并将其放置在工作台上（在新模块中以组件 2 建模）；物品加工完成后，第二台机器人将其放回传送带；这三个新组件共同替代人工操作员（见图 3）。

初始系统的算子与用于建模新增三个组件的算子进行复合，得到制造系统的新架构，之后便可在实际实施前对其进行分析。

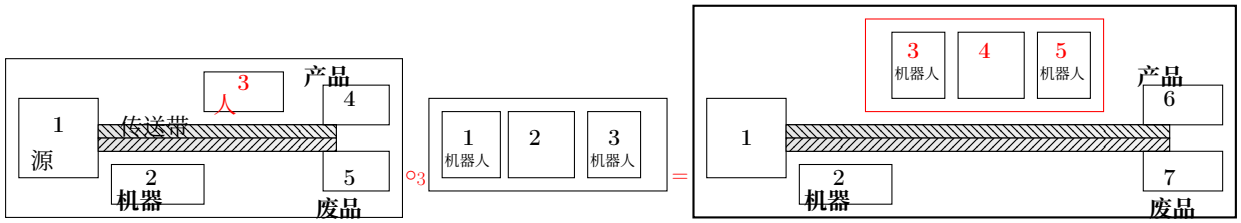

图 3：通过算子的复合对制造系统进行建模

算子代数。 代数算子 $P = (End_X, \circ_i)$ 上的一个代数由一个集合 X 和一个算子态射 $\mu: P \to End_X$ 给出。这意味着，对于每个算子 $f_n: X^n \to X$ of P^- with $n \geq 1$，我们都有一个态射 $\mu(f) = f$，使得 $f_n: X^n \to X$ 编码 f。

这里只提供了简短的背景介绍；有关算子及其进一步发展的研究，可参见 [6], [10], [5] 中的示例。

接下来，我们将使用 Event-B 构建一个通用模型，以对算子及其运算 $\circ_i$进行编码。请注意，与图 1 所示的算子相比，算子的图形化视图（见图 2）增加了更多细节。正如我们将在 Event-B 简介以及使用 Event-B 建模的部分中看到的，向抽象模型添加细节是通过精化 技术实现的。

2.2. 用于系统工程的 *Event-B*

Event-B [1, 4] 是一种建模与开发方法，其中组件被建模为上下文 和抽象机，这些组件组合并精化为称为精化的具体机。在 Event-B 建模过程中，抽象机构成动态部分，而上下文 用于描述静态部分。

一个上下文 由用作类型的载体集构成。它还可以包含常量（c）、假设性质（定义在集合和常量上，并用作公理）（Ax），以及由公理推导出的定理（Th）。上下文可以扩展以构建更大的上下文。建模项目中的机器会使用（引用）上下文。

一个抽象机描述了一个系统行为的数学模型，并将其视为离散转换系统。形式上，抽象机由一个状态空间描述，该状态空间由带类型变量（x）和不变式（$I(x, c)$）组成，此外还包括若干事件描述。它可以看到一个或多个预定义的上下文；也可以对其进行扩展以构建更大的机器。这里使用变量 x表示状态变量列表。

机器的状态空间。受不变式（类型谓词、性质）约束的变量描述了机器的状态空间。状态之间的转换由机器事件的作用引起。模型所要求的特定性质可以包含在不变式中。谓词 $I(x, c)$表示机器的不变式，其中 x 是状态变量列表， c 是已定义的常量。

抽象机的事件。在 Event-B 中，事件是对系统转换的描述。事件是自发的，并展示系统的演化方式。事件 e 建模为一个守卫替换： $e \triangleq (eG \Longrightarrow eB)$，其中 eG 是事件守卫， eB 是事件的主体或动作。其具体形式如下：

$$e \triangleq \textbf{任意}\ lv\ \textbf{满足}\ eG(x, lv, c)\ \textbf{则}\ eB(x, lv, c)\ \textbf{结束}。$$

lv 是该事件的局部变量（或参数）。为简化记号， c 在此处省略，记作 $eG(\cdots)$、 $eB(x)$、 $I(\cdots)$、 $BA(\cdots)$。

存在一个特定的初始化事件；它没有守卫，形式如下：

$$\text{init} \triangleq \textbf{开始}\ eB(x)\ \textbf{结束}$$

事件只有在其守卫成立时才能发生，即处于使能状态。若有多个事件处于使能状态，则进行非确定性选择。若没有事件处于使能状态，系统就会死锁。事件 e 的动作通过同时执行广义替换（S）来描述，当该事件发生时，

系统状态如何演变：此时，不相交的状态变量会同时更新。

替换 S 可以是确定性的，也可以是非确定性的。例如，基本的确定性替换[1] x := E(x,c)在逻辑上等价于前后谓词$x'such\,that\ \ x' = E(x,c)$[2]。其符号表示为 $x' := (x' = E(x,c))$，其中 x' 对应替换后的状态变量 x ，而 $E(x,c)$是一个表达式。非确定性替换表示为 $x :\in E$，其中 E 是一个表达式；也可以表示为 $x :\mid P(x,c)$，其中$P(x,c)$是一个谓词，用于刻画 x 的可能取值。

在 Event-B 中，定义证明义务，通过初始化建立不变式，并通过其他事件（会改变状态变量）保持不变式，从而确立模型一致性。系统的特定性质（包含在不变式中）也以同样的方式得到证明。形式上，证明义务的模式如下列 sequent 所示：

初始化证明义务	$BA_{\text{init}}(x') \vdash I(x')$
保持性证明义务	$I(x), eG(x,lv), BA_e(x,x') \vdash I(x')$

精化。 Event-B 方法的一个重要特点是可以利用精化技术，从抽象模型逐步进行扩充，设计出越来越具体的机器和系统。在精化过程中会引入新变量（y）；在不破坏抽象不变式$I(x,\ c)$的前提下强化不变式，最后强化事件守卫。在精化的 $J(x,\ y)$不变式中，抽象变量（x）与具体变量（y）相互关联。精化还伴有证明义务，用于证明其相对于所精化的抽象模型的正确性。抽象机器模型中具有前后谓词$BAA(x,\ x')$的事件，以及具体（精化后）模型中具有前后谓词 $BAC(y,\ y')$的事件，其证明义务为：

$$I(x) \wedge J(x,y) \wedge BAC(y,y') \implies \exists x'.(BAA(x,x') \wedge J(x',y'))$$

其直观含义是，具体事件可以由抽象事件模拟。

Rodin 工具。 Rodin[3] [2] 是一个开放且可扩展的工具，由多个专用于构建 B模型并对其进行推理的模块（插件）组成。Rodin 集成了多种定理证明器以及 ProB模型检查器 [7]。

3. 使用 Event-B 对算子进行建模

我们必须定义 Event-B 机器来对算子进行建模，其结构为 $O_X =$（结束 $_X(n)$ $\circ_i$），其中集合 X作为参数， n为自然数，结束 $_X(n)$ 是 $X^n \to X$的函数集合，并且 $\circ_i$ 由自然数集 i 参数化的复合函数。参数 X 是一个固定集合，将在上下文中定义；$End_X(n)$ 依赖于一个参数 n，其每个元素都是一个具有 n 个输入参数的算子，因此 $End_X(n)$ 将动态构造

[1]请注意，使用 ASCII 字体表示 Event-B 代码[2]x' 表示 x 动作之后的值[3]http://wiki.event-b.org/index.php/Main_Page

上下文 Ctx0 _Prm

SETS

算子 所有算子的集合——目前将 X 抽象为 NAT1

常量

最大参数数 最大输出数 最大算子数 最大后继数 序列自然数 Event-B 模型中使用的常量

公理

ax10: $finite(OPERAD)$ 算子是有限集合

ax15: $maxArgs \in \mathbb{N}_1$ 算子的最大参数数

ax20: $maxOut \in \mathbb{N}_1$ 算子的最大参数数

ax25: $maxOut = 1$ 用于仿真

ax30: $maxArgs \geq 6$ 至少为 1，此处为 6，用于仿真

ax35: $maxOprd \in \mathbb{N}_1$ 算子的最大值

ax40: $maxOprd \geq 8$ 至少为 1，此处为 8，用于仿真

ax45: $maxFol \in \mathbb{N}_1$ 花序的最大值 maxArgs*maxOprd

ax50: $maxArgs \leq maxFol$

ax55: $maxFol \geq (maxOprd * maxArgs)$

ax60: $SeqN \in \mathbb{N}_1 \pfun \mathbb{P}(\mathbb{N}_1)$ 生成一个包含 nn 个有序自然数的序列

ax65: $SeqN = (\lambda nn \cdot (nn \in \mathbb{N}_1 \land nn \geq 1) | \{kk \cdot kk \in \mathbb{N}_1 \land kk \leq nn | kk\})$

ax70: $\forall nn \cdot ((nn \in \mathbb{N}_1) \Rightarrow (\bigcup kk \cdot (kk \in 1 \,..\, nn) | \{kk\}) = SeqN(nn))$

END

图 4：建模的上下文

通过一个事件；i 是复合的一个参数，也将与两个算子一起动态定义，这两个算子是 o_i 的参数。

算子的复合涉及 Event-B 模型的演化，随后将通过一个事件来建模，即 composeSeq。为简洁起见，对于图 1 中的算子，我们采用记法 f(1，2，3，4)、g(1，2) 和 f(1，g(2，3)，4，5)。这是建模中的第一次抽象，因为我们暂时不使用集合 X，而是使用自然数。默认的输出参数也被忽略（将其抽象为 0，并简单地表示为一个点）。输入参数 1，2，3，4，$\cdots$ 在其算子中称为位置 或标签 。然后，我们在复合过程中对它们进行比较并重新标记。

3.1. 算子结构的编码

第2节中定义的算子结构部分编码在一个 Event-B 上下文中。

我们引入了一个上下文（见图4），用于定义必要的建模要素或参数，以及一个包含四个变量（我的算子、*arityOp*、叶集和*outOp*）和一个用于构建新算子的事件（newOperad）的抽象机，并通过另一个事件（composeSeq）复合两个给定的算子。

机器 算子　　_Abs 运算子抽象

变量

...

不变式

inv10: $myOperads \subseteq OPERAD$ // 所有已定义的算子

inv30: $arityOp \in myOperads \nrightarrow (1 \,..\, maxFol)$ // 元数（n, m, ...）

inv40: $foliage \in ((1 \,..\, maxFol) \leftrightarrow myOperads)$ // 各算子的叶集

inv60: $outOp \in myOperads \nrightarrow \mathbb{P}(1 \,..\, maxArgs)$ // 算子的输出

...

END

如第 2.1 节所述，任何算子 f 默认都具有给定数量（n）的输入和一个输出。输入是有序的，其位置用于算子的复合。为了使建模更具一般性并便于未来扩展，我们考虑函数 $outOp$，它可以定义多个输出，但在当前模型中受到约束。

最简单的情况是，将两个算子 $f(1..n)$ 和 $g(1..m)$ 复合，得到一个具有（$n+m-1$）个有序输入和 1 个输出的新算子。实际操作中，我们有一个算子集合（记作 $myOperads$），每加入一个新算子，该集合就会增长。粗略地说，变量$myOperads$ 对算子 End_X进行建模。

一般情况下，也就是在先前复合之后的任意位置进行复合，对复合操作进行建模会更加复杂。例如，考虑三个算子$f(1，2，3，4)$、$g(1，2，3)$ 和 $h(1，2，3)$，复合（$f \circ_2 g$）得到一个算子；若采用树表示（图 5），它有 6 个叶子；复合（（$f \circ_2 g$）$\circ_4 h$）则得到一个有 8 个叶子的算子（见图 6）。但是请注意，按照复合的位置（4），h 被插入到 g 中；例如，如果要在位置 5 进行复合，则会将其插入到 f 中：$(f \circ_2 g) \circ_5 h$。

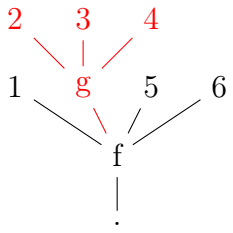

图5：复合（$f \circ_2 g$）

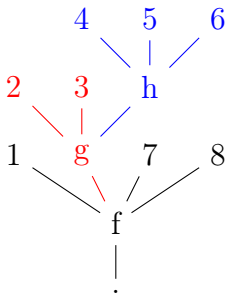

图6：复合（（$f \circ_2 g$）$\circ_4 h$）

因此，给定两个算子 op_1 和 op_2 以及 op_1中的一个位置 i，复合取决于参数 i，可以直接在 op_1 中完成（如果 i的值不超过 op_1的元数，且先前的复合尚未重构 op_1），也可以在先前插入的算子 op_1中完成。此外，所用算子的元数、算子参数的重新标记（例如在 $h(4，5，6)$中）以及最终叶子的数量，都不能直接帮助确定复合在哪个算子中完成。

为了驾驭这种复杂性，我们在算子及其复合的建模中引入了一些抽象：

- 一个算子有一个叶集：它包含所有输入位置（即叶子），无论处于复合的哪个层级

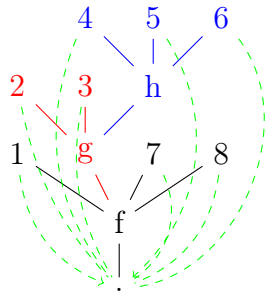

图7：复合 $((f \circ_2 g) \circ_4 h)$

；例如，$\{1,2,3,4,5,6,7,8\}$ 就是算子 $r = ((f \circ_2 g) \circ_4 h)$的叶集。

存在一个从 $\{1,2,3,4,5,6,7,8\}$ 到 r 的全函数（图 7 中以绿色显示）。叶集的基数大于或等于算子的元数。

- 复合之后，一个算子挂接在另一个算子中；例如在图 6 中，算子 h 挂接在算子 g中，算子 g 挂接在 f中。
- 给定位置 i 有一个帽歌剧； i直接位于一个算子中；例如，位置 3 的帽是算子 g，位置 5 的帽是算子 h。
- 复合到一个初等歌剧 op中之后，其位置或参数与 op之间的关系不再是全函数，因为有些位置被删除（用于复合的那些位置），而其他位置则被重新标记。这使得无法使用全函数将位置与其算子关联起来。

这些观察结果证明了我们用于对算子建模的结构是合理的（见图 8）；请注意，为了便于阅读并避免数据重复，我们仅给出精化的一个摘录。不过，开发的完整架构以综合的形式列于第3.4节。

3.2. 歌剧的复合建模

首先，我们引入一个事件，用于对初等（非复合）算子的构造进行建模。该事件的发生会生成一个初等算子集合，供后续复合使用。

构造一个新算子。 我们根据两个参数构建一个新算子：其输入数量 rr 和输出数量 vv。参数 rr 用于生成一个自然数序列 $(1..rr)$。事件 newOperad（见图9）对引入新算子的过程进行建模，例如 f(1，2，3，4)。该事件创建的新算子会被添加到我的算子 中，并可用于后续复合。请注意， $inOp$， $outOp$ 和 $myOperads$均已正确更新。为简洁起见，我们在此合并抽象机中的动作（标记为 a_i）和精化中的动作（标记为 ra_i）；守卫也同样处理（抽象机中的守卫标记为 g_i，精化中的守卫标记为 rg_j）。

机器 算子 _R1 用于定义算子的内部结构
细化 算子 _Abs
查看 Ctx0 _Prm
变量
———— 新变量 ————
inOp 我的算子中每个算子的输入
GHatOp 全局 HatOp（flor 上所有算子）
hookOp 一个算子挂接在另一个算子内部 {（内部 ⊢> 算子）}
gHookOp 全局钩子，一个算子中所有挂接的部分
不变式 ... :
———— 新不变式 ————
invr10: $inOp \in myOperads \pfun \mathbb{P}(1 \upto maxFol)$
算子的输入，按自然数序列组织
invr20: $GHatOp \in ((1 \upto maxFol) \times myOperads) \pfun myOperads$
给出任意复合歌剧代数中给定自然数（一个位置）对应的算子帽
invr30: $hookOp \in myOperads \pfun myOperads$ 是钩接算子
invr34: $dom(hookOp) \cap dom(outOp) = \varnothing$ 钩接算子没有输出
invr40: $gHookOp \in myOperads \pfun myOperads$ 钩接算子的闭包
invr50: $((myOperads \neq \varnothing) \Rightarrow$
$(\forall op \cdot ((op \in myOperads \land op \in dom(arityOp) \land op \in dom(inOp)) \Rightarrow$
$card(inOp(op)) \leq arityOp(op)))$
)
——— 随后是具体性质（SP1、SP2、SP3）———
...
END

图8：Event-B 中算子结构的摘录

算子的复合。 由于涉及多种情况以及参数算子树需要进行相应重组，对算子的复合进行建模颇具挑战性。经过分析并将其推广到任意两个算子 op_1 和 op_2（无论是基本算子还是由先前复合得到的算子）后，我们根据待复合的两个算子以及此处称为 ii 的复合位置，确定了四种不同情况。在运作所需的条件中，变量 ii 被视为 op_1叶集中的元素。算子 op_2 有其叶集，记作叶集*2*。

- 叶集中的位置 ii 直接位于算子 op_1 中（而非组成 op_1的内部算子中）；该位置 ii 将从 op_1的位置中删除。– 所有位于小于 ii 的位置上的参数都保留在复合结果的叶集中，不作任何更改。– 所有位于大于 ii的位置上的参数都保留在

事件 newOperad ⟨普通⟩ ≙ 构造一个具有 rr 个输入和 vv 个输出的算子

扩展 newOperad

any

新操作 rr vv seqrr seqvv

其中

g1: $card(myOperads) < maxOprd$

g3: $newOp \notin myOperads$

g4: $rr \in 1 \,..\, maxFol$ 算子输入数量

g6: $vv \in 1 \,..\, maxArgs$ 算子输出数量

g8: $seqrr = (1 \,..\, rr)$

g9: $seqvv = (\{1\})$ 默认值为 1，而不是 vv

g28: $card\,(foliage) + card\,((seqrr) \times \{newOp\}) \leq maxFol$

gr22: $finite((seqrr) \times \{newOp\})$

gr24: $rr = card((seqrr) \times \{newOp\})$

then

a1: $myOperads := myOperads \cup \{newOp\}$

a2: $outOp(newOp) := (seqvv)$

a3: $arityOp(newOp) := rr$

a4: $foliage := foliage \cup ((seqrr) \times \{newOp\})$ 该式随复合而增长

ra1: $inOp(newOp) := (seqrr)$ 输入参数的一个序列

ra3: $GHatOp := GHatOp \lhd (((seqrr) \times \{newOp\}) \times \{newOp\})$

end

图9：newOperad 事件的摘录

复合结果中，但应重新标记，将其当前标签增加 叶集*2* 的基数减 1。应根据新标签修改 op_1 中的输入位置。

- 位置 ii 间接属于一个参与复合的内部歌剧 op_1 （已挂接的歌剧 op_1）。

  – 所有小于 ii 的位置都保留在复合结果的叶集中，并且仍属于其初始帽歌剧，– 所有大于 ii 的位置都保留在复合结果的叶集中，但应通过将其当前标签加上叶集 *2* 的基数减1来重新标记。

在所有情况下，都应完全重新标记 op_2 的叶集，将 $ii-1$ 加到叶集*2*中的每个位置上，而不论它们在 op_2内属于哪个帽歌剧 。

为了进行建模，我们接下来需要确定帽歌剧，给定一个复合参数

ii，并相应地在 op_1 和 op_2中进行更新。我们还需要确定并修改挂接在其他算子中的算子，无论它们处于什么位置。

事件 ComposeSeq $\langle$**普通**$\rangle$ $\widehat{=}$ **将算子 op2 合并到算子 op1 的参数位置 ii 处，其中 op1、op2 和 ii ... op1、op2 是预定义的算子；ii 是 op1 叶集中的一个自然数，守卫：** $\cdots$ **图 11、图 12、图 13 展示了三个部分，然后 a1：** $outOp := (\{op2\} \lhd outOp)$ **删除 op2 的输出，只保留 op1 的输出 a2：** $foliage := ((foliage \rhd \{op1\}) \rhd \{op2\}) \cup (newfoliage1 \times \{op1\})$ **ra1：** $hookOp(op2) := hatopii$ **插入到帽 opii（op1 或其某个组成部分）中 ra2：** $gHookOp := (((gHookOp \lhd \{op2 \mapsto hatopii\}) \lhd (opHookinOp2 \times \{op1\}) \lhd (opHookinOp2 \times \{op2\}))$ **传播算子之间连接 op2 和 op1 的链接 ra3：** $GHatOp := (((GHatOp \rhd \{op1\}) \rhd \{op2\}) \lhd updGHatOp12)$ **帽的重构 ra4：** $inOp := (((\{op1\} \lhd inOp) \lhd liigiiupdinop1) \lhd updinop2)$ **结束**

图 10：事件 composeSeq 的摘录：动作部分

事件 composeSeq（见图 10）有若干参数，其中包括： $op1$、 ii、 $op2$。其结果是对算子 $op1$ 的修改，其中 op_2 的输出替换了 $op1$ 中位置 ii 处的输入，并且还进行了与复合相关的整体重构。

形式上，位置 ii 应位于 op_1 的叶集中，它首先用于确定实际执行复合的算子。为此使用关系 $GHatOp$ （见图 8，invr20）。每次复合后，都会在叶集中对所有大于 ii 的参数进行重新标记。

我们将事件 composeSeq 的守卫分为三个主要部分。第一部分引入所需的变量、类型检查约束，以及在局部变量中进行的一些中间计算；例如，已挂接到 $op1$ 中的算子，以便为守卫中的后续条件做准备（见图 11）。第二和第三部分守卫考虑对中间变量进行非确定性赋值，这些变量用于预先处理事件动作部分中状态变量的同步更改。我们给出守卫部分（经重新整理）的摘录（见图 11、图 12 和图 13）。

请注意，这里大量使用关系运算符，以帮助对已使用集合的变换和关系结构进行建模：算子 $\bigcup(\cdots)|\{\cdots\}$ 和 $A \cup B \cup \cdots$ 分别表示量化并和广义并； $s \lhd r$ 表示将关系 r 的定义域限制为集合 s； $r \rhd s$ 表示对关系 r的值域进行限制； $s \lhd\!\!\!- r$ 表示从关系 r的定义域中减去 s； $r \rhd\!\!\!- s$ 表示从 r的值域中减去 s。 $r_1 \lhd\!\!\!- r_2$ 通过左侧覆盖关系 r_1，其中 r_2。

composeSeq 事件的动作（见图 10）说明：(a1) $op2$ 从 $outOp$的定义域中删除，因为它现在用于复合；

xxx： .....rg20： $op1 \in dom(inOp)$rg22： $op2 \in dom(inOp)$ 它们具有输入参数rg26：
$op1 \notin dom(gHookOp)$rg24： $op2 \notin dom(gHookOp)$ 它们尚未在其他位置挂接rg40：
$opHookinOp1 = (\{op1\} \cup dom(gHookOp \rhd \{op1\}))$ 其中挂接了所有算子的 op1rg42：
$opHookinOp2 = (\{op2\} \cup dom(gHookOp \rhd \{op2\}))$

图11：事件 composeSeq 守卫的第 1 部分

(a2) 更新枝叶关系，以纳入与 $op1$ 相连的新叶集。(ra1) $op2$ 现在挂接在 $hatopii$ 中，即作为位置 ii 的帽所找到的算子；(ra2) 挂接在 $op2$ 中的算子现在传递地挂接在 $op1$ 中，从而建立全局关系；(ra3) 根据参数的新标签，更新各层级的帽算子；(ra4) 根据识别出的四种情况，更新 $op1$ 和 $op2$ 中的输入位置。

在守卫的第二部分（图 12）中：rg51 表示 $hatop1$ 是 $op1$ 中的所有帽算子；守卫 rg82 表示 $hatlii$ 是小于 ii 的位置 uu 的帽；标记为 rg102 的公式表示将 $op1$（的叶集）中大于 ii 的位置 kk 增加 $(kk + (cardfol2 - 1))$ 进行更新。所有这些守卫都通过守卫 rg119，为更新 op1 和 op2 中的帽作准备。

在守卫的第三部分（图 13）中，我们为算子 $op1$ 和 $op2$ 的输入参数 $(inOp)$ 更新准备条件和约束；在守卫 rg122 下，$luuupdinop1$ 表示小于 ii 的输入 kk 保持不变；在守卫 rg134 下，$giiupdinop1$ 表示大于 ii 的输入 kk 将被更新。随后将两者合并。$op2$ 的输入更新条件通过守卫 rg162 表示。

3.3. 涌现出的特定一致性属性

为确保复合算子（建模为一个事件）的一致性和正确性，我们引入了以下特定的一致性属性。

SP1:
$((myOperads \neq \varnothing \wedge (gHookOp \neq \varnothing) \wedge (GHatOp \neq \varnothing)) \Rightarrow (\forall op \cdot (op \in myOperads \wedge op \in ran($
$GHatOp) \wedge op \in dom(inOp) \wedge op \notin dom(gHookOp) \wedge (op \in ran(gHookOp)) \wedge op \in ran($
$foliage)) \Rightarrow ((dom(dom(GHatOp \rhd \{op\})) \cup inOp(op)) = dom(foliage \rhd \{op\}))))$ 叶集中的所有参数都带有帽

不变式（SP1）中表述的性质规定，叶集中出现的所有参数都具有一个帽歌剧；它们属于 $GHatOp$关系的值域，该关系将挂接到其他给定算子中的算子及其叶集联系起来。叶集中的参数满足这样的条件：它们传递性地位于这些算子的帽之下。

rg50: $hatop1 \in ((1 \mathinner{..} maxFol) \pfun myOperads)$ rg51: $hatop1 = foliage1 \domrest dom($ $GHatOp \rhd opHookinOp1)$ rg62: $hatopii \in opHookinOp1$ 可以是 op1，或是挂接在 op1 中的那些算子 rg64: $hatopii \in dom(inOp)$ rg70: $inOp(hatopii) \neq \varnothing$ hatopii 具有参数 rg72: ($ii \mapsto hatopii) \in hatop1$ 需要更新更大的 ii rg82: $hatlii = \{uu \cdot uu \in (1 \mathinner{..} maxFol)$ $\land uu \in foliage1 \land 1 \leq uu \land uu < ii | uu\} \domrest (hatop1 \rhd (opHookinOp1))$ rg92: $hatop2 = ($ $foliage2 \domrest dom(GHatOp \rhd opHookinOp2))\{op2\}$/ opHookinOp2 叶集2 rg94: $updhat2 = ($ $\bigcup oo \cdot (oo \in myOperads \land oo \in opHookinOp2) \mid (\bigcup kk \cdot (kk \in (1 \mathinner{..} maxFol) \land kk \in foliage2 \land ($ $kk \mapsto oo) \in hatop2 \land (kk + (ii - 1)) \in 1 \mathinner{..} maxFol) \mid \{(kk + (ii - 1)) \mapsto oo\}))$ rg102: $updgiihat = (\bigcup kk, oo \cdot (kk \in (1 \mathinner{..} maxFol) \land kk \in foliage1 \land (kk > ii) \land oo \in$ $myOperads \land (kk \mapsto oo) \in hatop1 \land (kk + (cardfol2 - 1)) \in 1 \mathinner{..} maxFol) \mid \{(kk + (cardfol2 - 1)) \mapsto$ $oo\})$ rg119: $updGHatOp12 = ((((hatlii \setminus (((ii + 1) \mathinner{..} cardfol1) \times \{op1\})) \cup (updgiihat))$ $\cup updhat2) \times \{op1\})$

图12：事件 composeSeq 的守卫第 2 部分

rg122: $liiupdinop1 = (\bigcup oo \cdot (oo \in myOperads \land oo \in dom(inOp) \land oo \in opHookinOp1) \mid \{oo \mapsto ($ $\bigcup kk \cdot (kk \in (1 \mathinner{..} maxFol) \land kk \in inOp(oo) \land kk < ii) \mid \{kk\})\})$ rg134: $giiupdinop1 = (\bigcup oo \cdot ($ $oo \in myOperads \land oo \in dom(inOp) \land oo \in opHookinOp1) \mid \{oo \mapsto (\bigcup kk \cdot (kk \in (1 \mathinner{..} maxFol)$ $\land kk \in inOp(oo) \land kk > ii \land ((kk + (cardfol2 - 1)) \in (1 \mathinner{..} maxFol))) \mid \{kk + (cardfol2 - 1)\})\})$ rg142: $liigiiupdinop1 = (\bigcup oo \cdot (oo \in myOperads \land oo \in dom((liiupdinop1) \cup (giiupdinop1))) \mid \{(oo \mapsto ($ $\bigcup args \cdot (args \in \mathbb{P}(1 \mathinner{..} maxFol) \land args \in ran(\{oo\} \domrest (liiupdinop1)) \cup ran(\{oo\} \domrest ($ $giiupdinop1))) \mid args))\})$ rg162:$updinop2 = (\bigcup oo \cdot (oo \in myOperads \land oo \in dom(inOp)$ $\land oo \in opHookinOp2) \mid \{oo \mapsto (\bigcup kk \cdot (kk \in (1 \mathinner{..} maxFol) \land kk \in inOp(oo) \land kk + (ii - 1) \in 1 \mathinner{..} maxFol)$ $\mid \{kk + (ii - 1)\})\})$

图13：事件 composeSeq 的守卫的第3部分：

SP2:
$(((myOperads \neq \varnothing) \land ((foliage) \neq \varnothing) \land (gHookOp \neq \varnothing)) \Rightarrow (\forall op \cdot ((op \in myOperads \land op \in ran($ $foliage) \land op \in ran(gHookOp) \land op \in dom(inOp)) \Rightarrow (dom(foliage \rhd \{op\}) = ((\bigcup oo \cdot ($ $oo \in myOperads \land oo \in (dom(gHookOp \rhd \{op\})) \land (oo \in dom(inOp)) \land (inOp(oo) \subseteq (1 \mathinner{..} maxFol)))$ $\mid (inOp(oo))) \cup (inOp(op)))))))$ 算子 op 的叶集中，输入参数来自挂接到 op 的算子

不变式 SP2 规定，算子 op 的叶集中的输入参数，即 $dom(foliage \rhd \{op\})$，仅来自挂接到 op 的算子。

SP3:
$(((myOperads \neq \varnothing) \land (inOp \neq \varnothing) \land (arityOp \neq \varnothing) \land (hookOp \neq \varnothing)) \Rightarrow \forall op \cdot ((op \in myOperads \land op \in dom(inOp) \land op \in dom(arityOp) \land op \in ran(hookOp) \land finite(hookOp \rhd \{op\})) \Rightarrow card(inOp(op)) = (arityOp(op) - card(hookOp \rhd \{op\})))$ 如果输入参数为空，也就是说，所有算子都已挂接在其中

最后，invariantSP3 规定，如果某个算子不再有任何输入参数，这是因为这些参数已被复合替换，替换它们的是其他算子（因此挂接在它们原来的位置）

3.4. Event-B 开发的架构

Event-B 开发的架构如图14所示。首先，我们对上下文 (Ctx0_Prm) 进行建模，其中汇集了对算子建模所需的基本集合和参数；然后通过 (Ctx1_Prm) 扩展该上下文，引入一个集合 X，以具体化输入/输出。抽象机 (Operads_Abs) 包含以下变量：$myOper\text{-}ads$ 一个算子集合，这些算子最终将被转换或复合；$outOp$ 每个算子的输出；$arityOp$ 每个算子的元数函数；以及$foliage$ 算子不断扩展的叶集。这些变量从抽象角度刻画算子的结构，此时只需要叶集；但我们尚未对叶集进行内部结构化。这将在精化中完成。事件 newOperad 和 composeSeq 定义在第一个抽象机 (Operads_Abs) 中。

模型的第一次精化（算子_*R1*）。在第一次精化中，我们引入了以下变量：$GHatOp$，一个关系，用于给出位置ii处任意给定参数的算子；$hookOp$ 一个偏函数，用于在存在时给出另一个算子所挂接的算子，以及$gHookOp$ 一个偏函数，用于给出在给定的另一个算子中挂接的所有算子。前述抽象事件也相应地进行了精化。

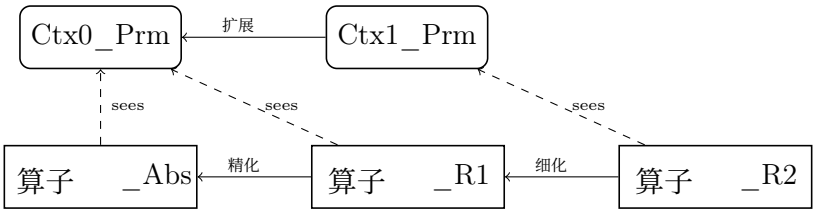

图 14：Event-B 开发的架构

模型的第二次精化（算子 *R2*）。这次精化用于引入集合X，该集合细化每个算子的输入和输出参数。

现在我们引入 X 来装饰算子的输入和输出。集合 X 可以看作是立方体或圆盘的一种抽象（见图 2），其中每个输入都有自己的 $x \in X$。这就像是在不改变抽象模型行为的情况下，为其添加细节。我们使用 Event-B 精化来对先前机器中添加这些细节进行建模；必要的结构已在先前的机器中引入，随后使其更加具体，添加一个

x到算子的每个输入/输出参数。例如，对于算子的输入参数（见图 8 中的 invr10），可通过部分单射（用算子 $\rightarrowtail$ 表示）来实现，它将一个 x 关联到每个输入参数：

$$inOpX : myOperads \nrightarrow (seqN(maxArgs) \rightarrowtail X)$$

为保证一致性，精化技术要求将 Event-B 模型的抽象结构与具体结构关联起来。例如，可以通过以下谓词 (1) 来实现，该谓词表示：抽象模型中所有具有输入参数的算子，在具体模型中也都具有参数：

$$\forall op.((op \in myOperads \wedge op \in dom(inOp)) \implies (inOp(op) = dom(inOpX(op)))) \quad (1)$$

通过抽象和精化，我们完成了一个 Event-B 模型的开发，该模型旨在为算子的使用提供一种正确性由构造保证的支持，从而实现其机械化。

4. 对所提出开发的评估与利用

我们已经共同 通过 Rodin/ProB）使用当前模式进行了若干实验 l.

4.1. 评估

如第 3 节所述，Rodin 平台为我们的开发提供支持。我们使用模型检验（借助 ProB）来补充 Rodin 各种证明器提供的定理证明。结合 Rodin 证明器，我们还充分利用 ProB 的功能 [8]（模型检验、动画、反证）来构建模型。我们对模型不变式以及特定性质 SP1、SP2、SP3 进行了全面分析。

Statistics × Goal Proof Information

Element Name	Total	Auto	Man.	Rev.	Und.
OperadsV6	**128**	**98**	**14**	**0**	**16**
Ctx0_Prm	1	1	0	0	0
Ctx1_Prm	0	0	0	0	0
Operads_Abs	29	28	1	0	0
Operads_R1	54	36	10	0	8
Operads_R2	44	33	3	0	8

图 15：使用 Rodin 进行证明的结果报告

证明统计。 图15展示了当前 Event-B 项目（使用 Rodin 实现的模型）的一些证明统计数据。抽象模型（名为 Operads_Abs）已全部证明。精化模型（Operads_R1 和 Operads_R2）中还剩 8 个证明义务。这些证明义务与基数证明或性质 SP_i 的表述中涉及区间的复杂类型检查有关。

我们检查了生成的动画轨迹，以分析可能发生的不变式违反，或发现并纠正模型中的错误。我们使用 ProB 检测不变式违反和无死锁性。图16展示了模型检验结果的统计报告。结果显示，新算子事件触发 45040 次、composeSeq 事件触发 128920 次后，没有违反任何不变式。观察到的死锁是指由于仿真值导致无法触发任何事件的情况（例如，为生成值而设定的集合基数受到限制）；这会导致重置机器参数，并继续进行分析。

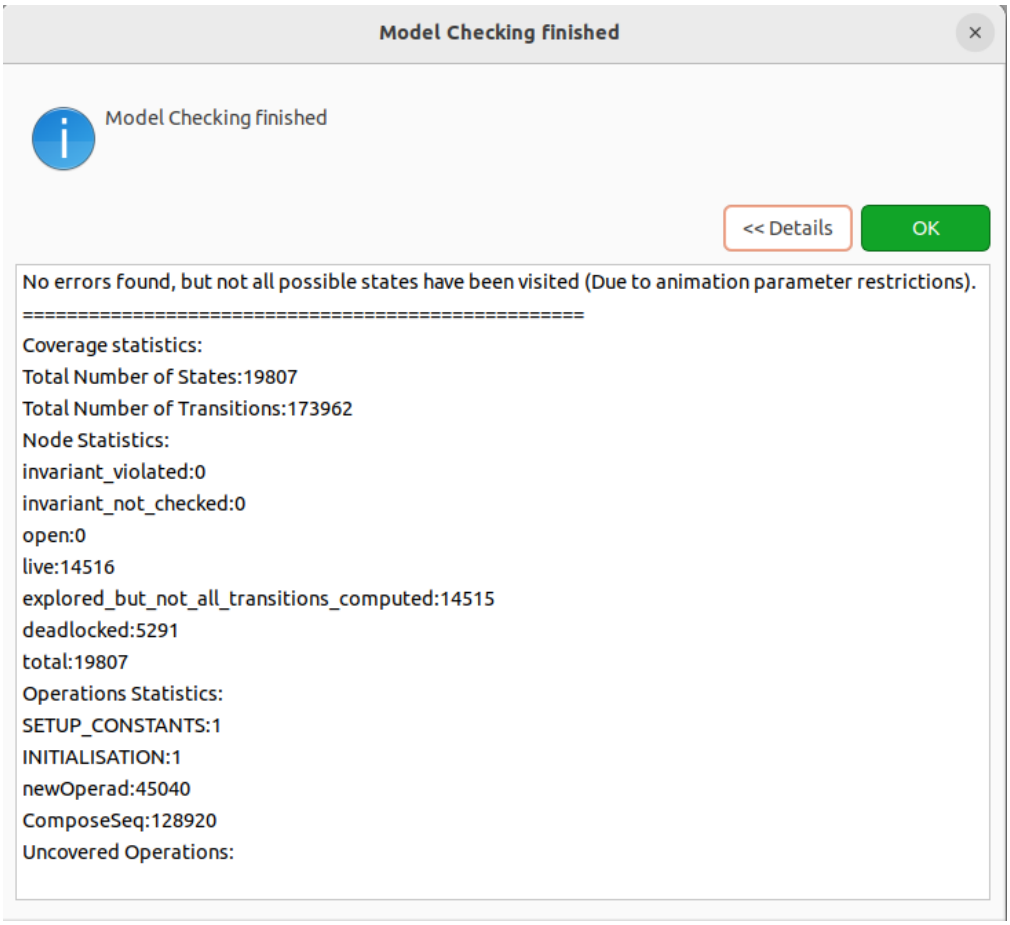

图 16：ProB 模型检验结果报告

4.2. 利用所提出的开发

我们构建的 Event-B 模型为操纵算子结构奠定了基础。借助 Rodin 和 ProB，可以轻松仿真算子的创建和顺序复合等基本运算。

仿真。 我们使用 Rodin+ProB 对所提出的 Event-B 模型进行了仿真，其中算子数量和输入参数数量均可任意指定。具体做法是在 Ctx0_Prm 上下文中实例化参数。

同样，我们模拟了算子的任意层级复合。模拟过程中还使用 ProB 对模型进行了模型检验，以分析所提出的算子机械化方法。

扩展。 当前的 Event-B 模型可以通过在 Operads_R1 机器或 Operads_R2 机器上应用 Event-B 精化技术，扩展算子上的其他运算（各种变换）。也可以定义特定性质，并将其添加到不变式中，以便按需扩展或约束模型。

可以通过精化来控制复杂系统中的异质性。例如，可以为复合算子及其参数定义特定的语义或行为，正如我们在 Operads_R2 精化中所做的那样。通常，集合 X 可以解释为或替换为一个语义域，例如向量空间。

同样，Event-B 模型可以作为语义基础和推理系统，用于在更高的抽象层次上对系统（自然系统或工业系统）进行建模。

5. 结论

我们探讨了算子的机械化，以推动利用基本对象的复合来建模和分析复杂系统。尽管已有多项理论研究涉及算子，本文则提出了算子代数的实用实施方案。这为具体的库实现奠定了基础，可用于符号计算或特定领域的应用。

研究结果总结。 我们构建了一个形式化 Event-B 模型，作为对算子进行符号计算的基础。该模型支持构建 n 元算子，并可在任意层级对其进行复合：用另一个算子替换指定位置上的参数，从而得到一个新的算子。Event-B 模型为此类代数结构的机械化提供了严谨的框架。这些结构有助于形式化实施各种应用，例如软件架构的组织或重组、复杂系统的描述与运行，以及各种演变中的动力系统的描述或仿真等。Event-B 提供的表达能力、抽象能力和精化技术，使我们能够以实用的方式处理这些算子结构。所提出的 Event-B 开发的结构可加以调整，以扩展当前工作，并应对其他具有挑战性的符号计算项目。

未来工作。 作为未来工作的一部分，我们计划构建算子的 Event-B 理论，以促进其在各个领域和应用中的广泛实际应用。首先，我们将通过添加交换性和结合关系来扩展当前提出的模型[6] ，并将它们定义为算子公理。这将有助于构建以算子建模的系统，并对这类系统进行推理。随后，可以通过预定义理论为树结构系统的各种构造和变换赋能并提供支持。

致谢。 这项工作源于我与同事 Johan L. 的富有成效的讨论。谢谢你，Johan！同事 Guillaume C. 给了我很多鼓励。谢谢你，Guillaume！

参考文献

[1] J-R. Abrial。《*Event-B* 建模：系统与软件工程》。剑桥大学出版社，2010年。[2] Jean-Raymond Abrial、Michael J. Butler、Stefan Hallerstede、Thai Son Hoang、Farhad Mehta 和 Laurent Voisin。Rodin：用于在 Event-B 中进行建模和推理的开放工具集。《*STTT*》，12(6):447–466，2010年。[3] John D. Foley、Spencer Breiner、Eswaran Subrahmanian 和 John M. Dusel。复杂系统设计规约、分析与综合的算子。《英国皇家学会会刊*A*辑：数学、物理与工程科学》，477(2250):20210099，2021年。[4] Thai Son Hoang、Hironobu Kuruma、David A. Basin 和 Jean-Raymond Abrial。在 Event-B 中开发拓扑发现。《*Sci. Comput. Program.*》，74(11-12):879–899，2009年。[5] Najib Idrissi。歌剧代数与同伦意义下的交换结构。《数学研究生期刊》，1:9–17，2016年。[6] Bruno Vallette Jean-Louis Loday。《代数算子》。数学科学基础丛书，346。施普林格，2012年。[7] M. Leuschel 和 M. Butler。ProB：B 的模型检查器。载于 A. Keijiro、G. Stefania 和 M. Dino 编，*FME 2003*：欧洲形式化方法，LNCS 2805，第855–874页。施普林格出版社，2003年。[8] Michael Leuschel 和 Michael J. Butler。Prob：B 的模型检查器。载于 Keijiro Araki、Stefania Gnesi 和 Dino Mandrioli 编，*FME 2003*：形式化方法，欧洲形式化方法国际研讨会，比萨，意大利，*2003*年*9*月*8*日至*14*日，会议录，计算机科学讲义第2805卷，第855–874页。施普林格，2003年。[9] Dev Sinha。小圆盘歌剧代数的同调，2010年。https://arxiv.org/abs/ math/0610236v3。[10] Jim Stasheff。什么是……歌剧代数。《美国数学学会会刊》，51，2004年6月。