arXiv:2601.04573v1 [cs.PL] 8 Jan 2026

部分指定状态的透镜（扩展版）

Kazutaka Matsuda[1][0000-0002-9747-4899](✉)，Minh Nguyen[2][0000-0003-3845-9928]，和 Meng Wang[2][0000-0001-7780-630X]

[1] 东北大学，日本仙台 980-8579。kztk@tohoku.ac.jp[2] 布里斯托大学，英国布里斯托 BS8 1TH。

摘要。双向变换是一对满足特定良好行为性属性的变换：一个将源数据映射到视图数据，另一个则将视图上的更改传回源。然而，当多个视图共享同一源时，对一个视图的更新可能会影响其他视图，因此很难在保留用户更新意图的同时维持对应关系，尤其是在多个视图同时发生更改时。在组合框架中确保这些属性则更具挑战性。本文提出了部分状态透镜，允许对源状态和视图状态进行部分指定，从而精确表示用户的更新意图。这些意图具有部分序关系，为合并来自多个视图的更新意图提供了明确的语义，并提出了与这种合并兼容的、更精细的更新保持概念。我们形式化定义了部分状态透镜，以及支持组合推理并确保更新保持的部分指定性知情的良好行为性。此外，我们通过示例展示了所提系统的实用性。

1 引言

双向变换提供了一种机制，通过将变换结果中的更改反向转换为原始形式，在多个数据表示之间传播更新。这使得各种数据结构之间能够实现同步，并有许多应用，包括经典视图更新 [4,7,17,22],打印器与解析器的协同开发 [31,35,51,56,57],支持编辑计算所得公式结果的电子表格 [10,29,50],能够从生成值中恢复选择的自反测试用例生成器 [16], 实时编程 [37,54,55],以及模型驱动工程中的模型同步 [45,52,53]。

尽管双向变换很有用，但开发双向变换仍需付出努力，因为程序员必须了解两个方向上的行为。此外，此类变换通常需要满足某些性质，才能被视为“双向”的。因此，人们提出了许多编程语言和框架来简化这项工作 [5,6,13,14,15,19,20,24,25,27,

30, 32, 33, 34, 36, 39, 47, 48, 49]。在这些系统中，最著名的是透镜框架 [13, 14]。

在透镜框架中，双向变换——或称透镜——由两个函数组成：$get\ \in S \to V$ 将源 状态 $s \in S$ 映射为其视图 状态$get\ \ s \in V$，而$put\ \in S \times V \to S$ 则结合 s 与更新后的视图 v'，返回更新后的源$put\ (s, v')$。由于get 并非总是单射，put 需要原始源 s，以重建get 变换过程中丢失的信息。由get/put 构成的一对函数称为良态（“双向”）透镜，前提是它满足以下两个性质。[3]

$$put\ (s, v') = s' \implies get\ s' = v' \qquad \text{(一致性)}$$
$$get\ s = v \implies put\ (s, v) = s \qquad \text{(可接受性)}$$

直观而言，一致性要求视图的任何更新都能通过put-再get得到保留。另一方面，可接受性确保如果视图没有更新，源也不会发生更新。透镜框架因其可组合的、正确性由构造保证的 方法而表现出色，使程序员能够通过复合更简单的透镜来构建复杂且行为良好的透镜，也使语言设计者能够扩展系统而不破坏这一保证。

自首次出现以来，透镜已通过各种方式得到扩展，但仍有若干研究挑战亟待解决。一个关键问题是如何处理多个视图。当多个视图共享同一个源，且用户更新其中一个视图时，我们希望相应调整其他视图，同时保留用户对直接修改的视图所做的原始更改。然而，允许这种行为会破坏传统的良好行为性属性，尤其是一致性。具体而言，用户提供的视图状态 v' 可能不同于同步后的最终状态$get\ (put\ (s, v'))$，因为对一个视图的更新（涉及 v'）会通过共享源传播到另一个视图。更一般地说，当多个视图同时被更新时，这些更新会通过源进行合并和传播，这意味着无法保证任何一个经过更新的视图保持不变。技术挑战包括：

– 对更新保持（针对一致性）和无更新（针对可接受性）的刻画，以及– 基于这些刻画的适当可组合的 良好行为性概念。

据我们所知，现有方法要么依赖全局（不可组合的）属性，要么采用不具备更新保持能力的较弱属性。例如，Hu 等人 [25] 和 Mu 等人 [39]提出了复制透镜，以支持具有所需行为的多个视图，但他们放弃了更新保留（一致性）。作为补救，Hidaka 等人 [19]提出了一种较弱的一致性属性，称为 WPutGet。在该属性下，修改后的视图和最终视图可以不同，但二者都必须对原始源产生相同的更新；

[3] 我们采用 Bancilhon 和 Spyratos [4]使用的术语。Foster 等人 [13, 14] 分别将可接受性定律和一致性定律称为 GetPut 和 PutGet。

这既不能保留用户提供的确切更新，也不具备可组合性。据我们所知，采用现有的透镜泛化或扩展并不能解决这一问题。许多这类方法（例如，[1,2,11,20,21]）要求更新后的视图必须与更新后的源所对应的视图完全相同，这一要求过于严格，无法容纳上述多个视图的行为。一些范畴论推广 [8,43]假定了 PutPut 定律 [14], 的一种形式，该定律直观地表明，put 必须保持更新组合。然而，在视图更新领域的文献中，这一定律被认为过于严格，无法支持实用的变换（参见例如 [14,26]）。

为了明确在多个视图共享同一个源时，更新是否得以保留，我们需要一个清晰的概念来表示用户更新背后的意图。考虑最简单的情形：一个源被直接复制到两个视图中。例如，如果源状态为 5，则对应的视图为 (5，5)。假设该视图被更改为 (7，5)。如果用户的意图只是将第一个视图从 5 改为 7，而不关心第二个视图，我们就可以安全地传播这项更新，得到更新后的源 7，其视图为 (7，7)，从而保留用户意图。相反，如果用户的意图是将这对视图更改为 (7，5)，那么就无法在保留用户意图的同时将更改传播到源。不过，如果我们将更新后的状态表示为 (7，5)，其意图就会变得模糊。一般而言，我们可能会在多个视图上进行意图各不相同的同步更新。在这种情况下，我们尝试将这些意图合并为一个。

这一观察自然地表明，可以用偏序来表示这些更新意图；比较两个更新意图 $u_1 \leq u_2$ 意味着u_1 被 u_2包含或保留。例如，在前面的例子中，对于任意 n，我们可以定义 $\Omega \leq n$，其中 Ω 表示一个未指定数值，而 (7，Ω) $\leq$ (7，7) 表示仅将第一个副本更新为 7 的意图在视图 (7，7) 中得到保留。此外，偏序为合并两个更新意图提供了一种自然的概念：最小上界，或称并（$\vee$）。一般而言，并运算符是偏函数，但这种偏性恰好适合用于建模冲突更新：回想一下，如果我们打算将源的一个副本更新为 7，另一个更新为 5，那么任何源更新都无法反映这些相互冲突的意图。

类似的思想也出现在无冲突复制数据类型（CRDT） [44],中。这类数据类型用于在无需协调的情况下，同步以独立且并发方式进行更新的分布式副本。在基于状态的 CRDT 中，副本状态构成一个并半格，其中（$\vee$）运算符用于合并更新。只要更新使状态大于或等于原始状态，并的交换性、结合性和幂等性就能保证每个副本都能在不发生回滚的情况下达到一致状态。发送完整状态似乎并不切实际，但已知只需发送增量状态 u，其中一次更新被描述为 $s \vee u$ [3]。尽管要求使用并半格和单调更新对于我们的目的而言通常过于严格，但这种联系表明，处理偏序域具有两项优势。首先，我们的双向变换现在可以处理 CRDT，从而扩大潜在的应用范围。其次，我们可以借鉴具体 CRDT 的一些思想，为具体的双向变换示例设计更新意图。

有了用于表示更新意图的偏序，剩下的挑战是建立一种适用于组合推理、能够感知序的良好行为性概念。良好行为性应当独立于部分指定状态的具体表示，以同时支持简单的部分指定状态（这类状态仅允许完全未指定）以及针对 CRDT 等具体数据类型定制的部分指定状态。

本文提出了一种新颖的双向变换框架，称为部分状态透镜。其关键特征在于，透镜能够处理偏序的更新意图，我们称之为部分指定状态。借助偏序，我们的框架提供了可组合的良态行为，可保证在共享同一源的多个视图之间保留更新，且不受部分指定状态具体表示的影响。具体而言，我们作出以下贡献。

– 我们通过一个具体场景阐释核心思想（第 2 节）。在借此场景说明共享源问题后，我们指出经典透镜的一个局限：缺少能够表达用户意图的域，也就是表达某项更新最终应产生何种结果的意图。随后，我们展示部分指定状态如何解决这一问题。– 我们对部分状态透镜框架进行了形式化，该框架纳入了关于部分指定性的可组合良态行为（第 3 节）。这是首个同时实现以下所有目标的透镜框架：(1) 通过共享源在多个视图之间传播视图到视图的更新；(2) 保证保留用户更新；(3) 组合推理；以及 (4) 与经典透镜框架互操作 [13, 14]。– 我们通过重新考察第 2 节中的场景及其非正式解决方案，展示所提框架的实用性，并说明如何通过具体透镜的复合对该场景建模（第 4 节）。– 我们给出一种将更新编码为部分指定状态的方法，由此证明部分状态透镜是基于状态的系统与基于操作的系统的混合体 [2, 11, 21]（第 5 节）。

最后，我们将讨论相关工作（第6节），并总结本文（第7节）。部分证明见附录（附录A）。我们还提供了 Agda 中的机械化证明[4] 以及 Haskell 中的原型实现。[5]

2 动机示例：共享源问题

本节将通过一个动机示例，概述我们的方法。

2.1 一个动机情景

考虑使用两个视图来管理任务（待办事项）：一个视图（进行中）显示所有进行中的任务，另一个视图（今天）显示所有今日到期的任务。我们假设这些任务

[4] https://github.com/kztk-m/ps-lenses-agda [5] https://github.com/kztk-m/ps-lenses-hs



图1：复合透镜的正向（*get*）行为 ***dup*** 和筛选 $_{\neg\text{已完成}}$/筛选$_{\text{Apr1}}$

可以单独更新，可能是因为这些视图对应于同一应用程序中的不同窗格，也可能是因为这些任务与他人共享。本例展示了一种使用双向变换提取（并转换）数据、供应用程序组件管理的情形。这是 Web 应用程序编程中常见的入门示例。[6]我们还可以进一步提取更小组件所需的更小数据片段（例如文本框中的任务名称），这样会更理想；但为简单起见，这里我们考虑这两个粒度较粗的视图。

例如，对于以下整体任务集合，

$$s_{\text{tl}} = \left(\begin{array}{l|lll} \text{ID} & \text{Done} & \text{Name} & \text{Due} \\ \hline 001 & \text{False} & \text{Buy milk} & \text{Apr 2} \\ 002 & \text{True} & \text{Walk dog} & \text{Apr 1} \\ 003 & \text{False} & \text{Jog} & \text{Apr 1} \end{array}\right)$$

其对应的视图为 v_{og} 和 v_{dt} （假设今天是4月1日）：

$$v_{\text{og}} = \left(\begin{array}{l|lll} \text{ID} & \text{Done} & \text{Name} & \text{Due} \\ \hline 001 & \text{False} & \text{Buy milk} & \text{Apr 2} \\ 003 & \text{False} & \text{Jog} & \text{Apr 1} \end{array}\right) \qquad v_{\text{dt}} = \left(\begin{array}{l|lll} \text{ID} & \text{Done} & \text{Name} & \text{Due} \\ \hline 002 & \text{True} & \text{Walk dog} & \text{Apr 1} \\ 003 & \text{False} & \text{Jog} & \text{Apr 1} \end{array}\right)$$

（我们使用类似表格的记法仅为便于阅读；此处不对其实际表示方式作任何假设。）

接下来，我们考虑通过双向变换将源与两个视图连接起来。假设从整个源到这些视图的连接分别由 筛选$_{\neg\text{Done}}$ 和 筛选$_{\text{Apr1}}$ 给出。我们不再分别考虑它们，而是可以使用复制透镜 [25, 39] 将它们捆绑起来。因此，整个变换由 *dup* 与 筛选$_{\neg\text{Done}}$/筛选$_{\text{Apr1}}$ 的复合给出，其前向行为如图1所示。在前向方向上，*dup* 只是复制源，使得待复合的透镜可以作用于它们各自的副本。在后向方向上，*dup* 合并这些副本上的更新——具体定义暂且留待后文，因为如何为组合推理提供一个合适的定义，正是本节讨论的主题之一。从理论上讲，*dup* 使我们能够专注于单个透镜，而不必考虑透镜之间的交互。从实践上讲，得益于这种内化，我们可以在变换中动态引入共享，

[6] 例如，https://todomvc.com/，不过视图有所不同。

从而能够复合涉及共享的透镜。类似 dup 的操作对于为（非线性的）双向编程语言 [19, 36] 提供可组合的语义尤其有用。

2.2 问题

考虑向进行中视图添加一个新任务“买鸡蛋”，如下所示。

$$v'_{\mathrm{og}} = \left(\begin{array}{l|lll} \mathrm{ID} & \mathrm{Done} & \mathrm{Name} & \mathrm{Due} \\ \hline 001 & \mathsf{False} & \text{Buy milk} & \text{Apr 2} \\ 003 & \mathsf{False} & \text{Jog} & \text{Apr 1} \\ 004 & \mathsf{False} & \text{Buy egg} & \text{Apr 1} \end{array}\right)$$

通过反向变换（put）以及随后的正向变换（get）传播此更新后，预期添加的元组会出现在原始源和今日视图中，如下所示：

$$s'_{\mathrm{tl}} = \left(\begin{array}{l|lll} \mathrm{ID} & \mathrm{Done} & \mathrm{Name} & \mathrm{Due} \\ \hline 001 & \mathsf{False} & \text{Buy milk} & \text{Apr 2} \\ 002 & \mathsf{True} & \text{Walk dog} & \text{Apr 1} \\ 003 & \mathsf{False} & \text{Jog} & \text{Apr 1} \\ 004 & \mathsf{False} & \text{Buy egg} & \text{Apr 1} \end{array}\right) \qquad v'_{\mathrm{dt}} = \left(\begin{array}{l|lll} \mathrm{ID} & \mathrm{Done} & \mathrm{Name} & \mathrm{Due} \\ \hline 002 & \mathsf{True} & \text{Walk dog} & \text{Apr 1} \\ 003 & \mathsf{False} & \text{Jog} & \text{Apr 1} \\ 004 & \mathsf{False} & \text{Buy egg} & \text{Apr 1} \end{array}\right).$$

为实现这种行为，dup 的反向行为必须接收 $(s'_{\mathrm{tl}}, s_{\mathrm{tl}})$ 以及原始源 s_{tl}，并返回 s'_{tl}。现有的 dup 透镜通过将副本与原始源 [25] 进行比较来检测哪些副本已更新，或者显式标记哪个视图已更新 [39]，从而实现这一点。然而，这种方法存在两个问题。首先，这种特定行为违反了第 1 节中提出的一致性属性 [4, 14]，即要求<跨?>更新保留</跨?>；我们将视图更新为 $(v'_{\mathrm{og}}, v_{\mathrm{dt}})$，但在执行 put 后接 get，最终得到 $(v'_{\mathrm{og}}, v'_{\mathrm{dt}})$。据我们所知，目前不存在既可组合、又允许通过共享源进行此类视图间更新传播的更新保持概念。其次，这种方法无法扩展到两个视图同时更新的情况。同时更新还带来另一个难题：即使只关注单个视图，一般情况下也无法再满足一致性属性。即便没有同时更新，只关注单个视图来表述一致性也并不理想，因为这会使推理变得复杂：我们需要进行别名分析，以防止变换中混淆副本，否则就无法从最终结果中投影出“一个视图”。

2.3 方法：部分状态透镜

上述问题源于仅凭状态不足以表达用户意图。例如，如果我们能够说明视图（v'_{og}，v_{dt}）中的变更是在第一个视图中插入任务 ID 004，那么我们就可以将更新保持表述为：插入的任务在同步状态（v'_{og}，v'_{dt}）之后仍然存在。要表述更新保持，这类意图

必须进行比较；要支持同时更新，还必须配有合并操作。这表明这类意图必须是偏序的。

我们将这些意图称为部分指定状态，并让我们的透镜除了处理普通的全定义状态外，也能处理它们。可以将部分指定状态视为状态与更新（也称为差量 [11],编辑 [21]或操作 [38, 44]）的混合体。与更新一样，它们能够表达更细致的意图——例如，区分插入单个任务与整个状态变更；而作为状态，它们则可以通过使用偏序进行比较，简洁地表述更新保持。差量状态 CRDT [3],也采用了类似思路：它们同样具有偏序，并兼具差量与状态的优点，其合并操作保证是全定义的——但这对我们而言过于强，因此不适用于我们的目的。与无冲突复制数据类型不同，后者的分布式副本无需协调即可异步更新，而put 的执行通常是经过协调且同步的，因此失败是一个可行的选项；相比之下，竞争性更新会被静默解决。

让我们回到第 2.1 节的动机示例。关于此示例中使用的部分指定状态的形式化定义，我们留待第 4 节讨论；这里先说明部分指定状态的直观行为。当用户将编号为 004、名称为“买鸡蛋”、截止日期为 Apr1 的任务添加到进行中视图，并离开今日视图时，如果更新背后的用户意图是让新增任务保留在进行中视图中，那么这一意图就表示为一对 $(w_{\mathrm{og}},\ \Omega)$，其中

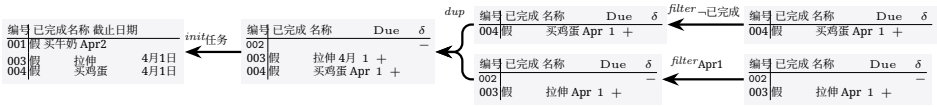
$$w_{\mathrm{og}} = \left(\begin{array}{l|llll} \mathrm{ID} & \mathrm{Done} & \mathrm{Name} & \mathrm{Due} & \delta \\ \hline 004 & \mathsf{False} & \mathrm{Buy\ egg} & \mathrm{Apr\ 1} & + \end{array}\right).$$

为便于直观呈现，我们将（严格）部分指定状态表示为带有一个额外字段 δ（用于记录更新标记）的表。(+) 标记表示相应元组必须存在；也就是说，它请求插入或更新带标记的元组。我们还有 (–) 标记，表示相应元组不得存在；带有 (–) 标记的元组只包含标识符，因为这些信息足以执行删除。形式上，这些意图通过部分指定状态与完整状态之间的序来建模。严格部分指定状态按子集包含关系排序，最小元记为 Ω，[7]，它对应于空表，直观上表示“任意内容”。

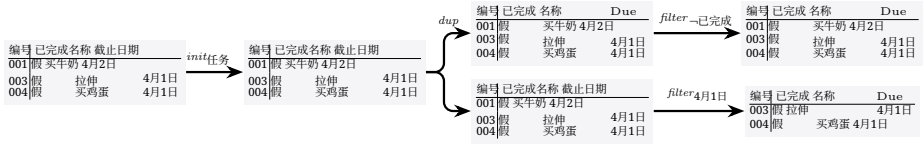
筛选$_{\neg\mathrm{Done}}$ 和筛选$_{\mathrm{Apr1}}$ 的反向行为只是将 w_{og} 和Ω 直接传递过去。随后，使用 $\vee$ 作为 $w_{\mathrm{og}} \vee \Omega = w_{\mathrm{og}}$，通过$dup$ 将它们合并；请注意，Ω 是最小元，也就是 $\vee$的单位元。这个结果表明，为了适应视图更新（w_{og}，Ω），应将 w_{og} 中标记为（+）的任务插入 s_{tl}（从而得到 s'_{tl}）。

我们可以在此结束反向变换，假设某个外部过程利用部分指定状态 w_{og} 的更新特性，将 w_{og} 反映到 s_{tl}。相反，类似于将源共享内化的 dup，我们也在框架中将这一更新过程内化为一个透镜。这不仅简化了我们的形式化过程，还使得此类更新反映的

[7] 为避免与通常用于表示偏函数未定义性的 $\perp$ 混淆，我们使用 Ω 表示部分指定状态中的最小元（如果存在）。



(a) 反向 (put)，重用图1中的原始源；所有涉及的透镜都是合法的。



(b) 图2a 中反向执行后向前执行（get）

图2：部分状态透镜的复合：dup 和筛选$_{\neg\mathrm{Done}}$/筛选$_{\mathrm{Apr1}}$，用于同步更新

组合式设计成为可能。具体而言，我们使用一种称为 ps-发起器的透镜，其 get 操作将完整状态嵌入部分指定状态，而其 put 操作则尝试将部分指定状态反映回原始完整状态，从而实现部分指定状态作为更新的语义。对于此示例，我们将一个特定的 ps-发起器记作 $\mathsf{init}_{\mathrm{tasks}}$，其 put 操作的设计是将例如 $(s_{\mathrm{tl}}, w_{\mathrm{og}})$ 转换为 $s'_{\mathrm{tl}\circ}$。一般而言，对于 (s, w)，它的 put 操作会将 w 中标记为 (+) 的任务更新或插入到 s 中，具体取决于 s 中是否已存在具有相同编号的任务（也就是说，它的 put 操作会将 w 中标记为 (+) 的任务插入或更新到 s 中），然后删除由 (–) 标记的任务。$\mathsf{init}_{\mathrm{tasks}}$ 的 put 操作遵循上述部分指定状态的含义；形式上，它的 put 定义满足：对于 s'，若 $\mathsf{put}\,(s, w) = s'$，则 $w \leq s'$ 成立，特别是 $w_{\mathrm{og}} \leq s'_{\mathrm{tl}\circ}$。现在，源已更新为 $s'_{\mathrm{tl}\circ}$。s'_{tl} 的视图为 $(v'_{\mathrm{og}}, v'_{\mathrm{dt}})$。基于前面解释的原因，我们有 $w_{\mathrm{og}} \leq v'_{\mathrm{og}}$；又因为 Ω 是最小元，所以有 $\Omega \leq v'_{\mathrm{dt}}$。于是，根据逐点序，$(w_{\mathrm{og}}, \Omega) \leq (v'_{\mathrm{og}}, v'_{\mathrm{dt}})$，这表明更新意图 $(w_{\mathrm{og}}, \Omega)$ 实际上得到了保留。

这种方法很容易扩展到同步更新。假设用户在今日视图中删除编号为 002 的任务，并将编号为 003 的任务名称更改为“拉伸”，同时在进行中视图中执行更新 w_{og}。另假设该更新的底层意图恰好是删除和更改任务名称。那么，我们可以将用户意图表示为（w_{og}，w_{dt}），其中

$$w_{\mathrm{dt}} = \left(\begin{array}{l|lll} \mathrm{ID} & \mathrm{Done} & \mathrm{Name} & \mathrm{Due} \quad \delta \\ \hline 002 & & & \quad - \\ 003 & \mathsf{False} & \mathsf{Stretch} & \mathrm{Apr}\ 1\ + \end{array}\right).$$

同样，筛选$_{\neg\mathrm{Done}}$ 和 筛选$_{\mathrm{Apr1}}$ 的反向行为不执行任何有意义的操作，并原样返回（w_{og}，w_{dt}）。然后，$put\ dup$ 会将两次更新合并为：

$$w_{\mathrm{merged}} = w_{\mathrm{og}} \vee w_{\mathrm{dt}} = \left(\begin{array}{l|lll} \mathrm{ID} & \mathrm{Done} & \mathrm{Name} & \mathrm{Due} \quad \delta \\ \hline 002 & & & \quad - \\ 003 & \mathsf{False} & \mathsf{Stretch} & \mathrm{Apr}\ 1\ + \\ 004 & \mathsf{False} & \mathsf{Buy\ egg} & \mathrm{Apr}\ 1\ + \end{array}\right)$$

之后，$put\ init_{\text{tasks}}$ 反映到 s_{tl}中，结果为：

$$s''_{\text{tl}} = \left(\begin{array}{l|lll} \text{ID} & \text{Done} & \text{Name} & \text{Due} \\ \hline 001 & \mathsf{False} & \text{Buy milk} & \text{Apr 2} \\ 003 & \mathsf{False} & \text{Stretch} & \text{Apr 1} \\ 004 & \mathsf{False} & \text{Buy egg} & \text{Apr 1} \end{array}\right)$$

因此，通过正向变换，视图变为：

$$v''_{\text{og}} = \left(\begin{array}{l|lll} \text{ID} & \text{Done} & \text{Name} & \text{Due} \\ \hline 001 & \mathsf{False} & \text{Buy milk} & \text{Apr 2} \\ 003 & \mathsf{False} & \text{Stretch} & \text{Apr 1} \\ 004 & \mathsf{False} & \text{Buy egg} & \text{Apr 1} \end{array}\right) \quad v''_{\text{dt}} = \left(\begin{array}{l|lll} \text{ID} & \text{Done} & \text{Name} & \text{Due} \\ \hline 003 & \mathsf{False} & \text{Stretch} & \text{Apr 1} \\ 004 & \mathsf{False} & \text{Buy egg} & \text{Apr 1} \end{array}\right)$$

这种反向和正向行为如图2所示。再次注意，w_{og} 和 w_{dt} 分别在 v''_{og} 和 v''_{dt}中得到保留。

2.4 另一项优势：对更新的细粒度描述

正如我们之前提到的，部分指定状态介于（完整）状态和更新之间。这种双重特性使我们的基于状态的框架能够享有基于操作的系统 [2,11,21,38]的一些优势。我们的 put 可以区分对状态产生相同净效果的更新，因为部分指定状态能够传达从完整状态中无法看到的信息。

回顾一下，动机示例对待办事项管理应用中所管理的数据提取进行了建模。因此，理想情况下，视图应支持任务插入、任务重命名、任务完成和任务延期等操作。然而，众所周知，在纯基于状态的系统中实现这一点十分困难。例如，在类似经典透镜 [14], 的基于状态的系统中，对于进行中视图，我们无法区分任务完成和任务删除，因为它们对状态产生的净效果相同；任务完成和任务删除都会将任务从进行中视图中移除。同样，对于今日视图，出于相同原因，我们无法区分任务延期和任务删除。区分此类更新的需求，是基于操作的系统 [11,21,38],的一项关键动机；在这类系统中，put变换的是更新（或操作），而不是状态。

部分指定状态也可以通过扩展部分指定状态来应对这一特定挑战。对于进行中视图，或 筛选$_{\neg\text{Done}}$ 的视图，我们还考虑标记（✓），它直观上要求完成任务；但在 筛选$_{\neg\text{Done}}$ 的视图中，每项任务都处于进行中——

ID	Done	Name	Due	δ
001				−
003	True	Jog	Apr 1	✓

图 3：涉及完成和删除的部分指定状态

.....，它们只能导致移除带标记的任务。图 3 展示了我们新的部分指定状态的一个示例。put 筛选$_{\neg\text{Done}}$的行为会将（✓）标记转换为（+）标记，以反映（✓）标记的意图；请记住，（+）标记表示插入或更新。$init_{\text{tasks}}$ 和dup 的行为没有变化。例如，对于原始源 s_{tl} 和更新后的视图，

图 3 中，$put\ init_{\text{tasks}}$ 会返回以下任务，其中编号为001的任务已删除，编号为003的任务已完成。

ID	Done	Name	Due
002	True	Walk dog	Apr 1
003	True	Jog	Apr 1

同样，对于今日视图，我们考虑标记 (P)，其直观含义是推迟；该标记不传达推迟天数的信息，因为这类信息已包含在任务本身中。与 (✓) 的情况类似，带有 (P) 标记的元组表示这些元组不会显示在 筛选$_{\text{Apr1}}$的视图中。同样，在 put 执行过程中，筛选$_{\text{Apr1}}$ 会将 (P) 转换为 (+)，以请求推迟。

需要注意的是，将这类部分指定状态与完整状态进行比较时，我们无法避免二者在表达用户意图方面存在差距。例如，在 筛选$_{\neg\text{Done}}$的视图中，由于该视图不允许出现已完成任务，我们无法通过完整状态区分删除与完成。这意味着，在该视图中，$w \leq t$ 对于完成意图 w 以及完整状态 t，其中待完成的任务在 w 中不存在。这是固有的：在 筛选 $_{\neg\text{Done}}$ 的视图中，完成与删除之间的区别无法用完整状态表达。然而，这并不直接意味着我们的透镜会混淆这类意图。我们通过设计加以避免：也允许 get采用部分指定状态，这还带来了其他好处（第 2.5 节）。

2.5 get 中的部分指定状态

在第 2.3 节中，我们主要关注 put 如何利用部分指定状态，如图 2a 所示。在讨论良好行为性的组合推理时，get 方向就变得相关了；这涉及关于 put-然后-get 和 get-然后-put行为的论述。

由于部分指定状态的主要用途是改善 put处理多个视图时的行为，因此自然会问，为什么我们还关心它们在 get中的作用。这种设计兼具理论和实际优势。

简单性是最显而易见的理论优势之一。如果 get 只处理完整状态，而 put 只传播部分指定状态，那么原始状态和更新后的状态将属于不同的域，正如更新-更新透镜中 $get \in S \to V$ 和$put \in S \times P \to Q$ 的情况类似；其中 P和 Q 分别是与完整状态集合 S 和 V 对应的部分指定状态集合。然而，这会使良好行为性（往返性质）变得复杂，因为我们无法将put 的结果直接传递给get，反之亦然。我们可能需要建立连接，将 S 连接到 P，并将 V连接到 Q，而透镜也必须保持这些连接。

这种简洁性带来了兼具理论和实践意义的优势：可复用性。我们的透镜在计算层面遵循经典透镜 [13, 14] ：在我们的框架中，S 与 V之间的透镜是一对变换$get \in S \to V$和$put \in S \times V \to S$，它们作用于相同的定义域（暂且忽略这些变换的部分性）。唯一的区别是，在推理透镜的良好行为性时，我们考虑 S 和 V 中的部分指定状态及偏序。得益于这种设计，我们可以复用许多透镜和

现有文献中的透镜组合子，不过需要承担证明我们版本的良好行为性的义务。透镜复合就是这样一个例子。在一种特殊情况下，经典透镜框架 [13, 14] 中的良态透镜在我们的框架中也属于良态透镜。另一个优势是，我们可以为自己的透镜复用现有的透镜表示。例如，我们可以对变换进行元组化 [9, 23]，将源的常见计算打包到get 和put [46]中。我们也可以使用范拉尔霍芬透镜 [28, 40]（如 Haskell 的 lens 库中所用），或 Profunctor 光学 [42]。

这种保证强度是另一项重要优势。如果我们将get限制为只能作用于完整状态，那么更新保持只能通过比较部分指定状态与完整状态来表述，因此难以排除那些会混淆更细致意图的透镜；这类意图只能用部分指定状态表示，例如在put中将标记为(✓)的元组替换为标记为(–)的元组。如果此类透镜的get 也适用于部分指定状态，尤其是适用于put 返回的部分指定状态，我们就可以通过比较更新后的视图与更新后的源的视图，发现put 未能保持用户意图。

在get 中使用部分指定状态的另一项实际优势，是能够以一定的优先级处理多个视图上的竞争更新；在这种情况下，我们的透镜将处理（增量状态 [3]）无冲突复制数据类型 [44], 。粗略地说，这类数据类型的状态是偏序的，其中 $\vee$ 是全定义的（也就是说，put 对dup 永远不会失败）。

作为这些优势的交换，我们的框架要求程序员为部分指定状态定义 get 。这不会造成过重负担，因为我们将 $put\ (s,\ v)$ 定义为返回一个部分指定状态，并确保其中的信息足以让 get 为 v 返回有意义的结果。

3 部分指定状态的透镜

本节作为本文的主要贡献，将介绍我们提出的透镜框架——部分状态透镜（简称 ps-lenses），它能够处理第 2 节中 v_{og} 和 v_{dt} 这类未指定状态。我们形式化工作的关键在于，它支持组合推理，与原始透镜框架 [13,14] 类似。

3.1 预备知识：经典透镜

我们首先回顾经典透镜框架 [13, 14], ，我们的框架以此为基础。如第 2.5 节所述，主要区别在于定义域上的定律和结构。为简化起见，并且由于 put 可能因各种原因而失败（例如副本上的冲突更新），本文考虑集合之间的偏函数，而非完全偏序（CPO）。尽管这实际上排除了通用递归，只允许终止函数，但我们认为，将讨论直接扩展到完全偏序是可行的。

一个“源”集合 S 与“视图”集合 V之间的（偏）（非对称）经典透镜 ℓ 由一对偏函数 $get \in S \to V$和 $put \in (S \times V) \rightharpoonup S$构成。这里，我们用 $\rightharpoonup$ 强调 put 可能是偏函数，而为简单起见，我们假设 get是全定义的。我们用 Lens $S\ V$ 表示所有透镜 $\ell =$ （get,put）的集合，它们定义在

S 和 V之间，并用 $get\ \ell$ 和 $put\ \ell$ 表示它们的第一和第二个组件。直观来说，$get\ \ell \in S \to V$从源 s中获取视图；而 $put\ \ell \in (S \times V) \rightharpoonup S$会将视图更新传播为源更新：前者将 $get\ \ell\ s$ 更改为某个 v'，后者则进而将 s 更改为 $put\ \ell\ (s, v')$。由于 put 可能是偏函数，更新传播可能会失败。

并非所有透镜都合理。以下定律确立了透镜的双向性 [4, 14, 17]。为了给后续讨论奠定基础，我们还为这些定律配上了非正式属性（P），说明它们所保证的性质；我们将力求让部分状态透镜也满足这些属性。

$$\forall s, s' \in S, \forall v' \in V.\ put\ \ell\ (s, v') = s' \implies get\ \ell\ s' = v' \qquad \text{(consistency)}$$

“对源的视图所做的更新会保留在更新后的源中。” (P1)

$$\forall s \in S, \forall v \in V.\ get\ \ell\ s = v \implies put\ \ell\ (s, v) = s \qquad \text{(acceptability)}$$

“如果没有更新源的视图，就意味着源没有更新。” (P2)

直观地说，一致性属性（也称为 PutGet [14], ）指出，更新后的源 s' （由put返回）会正确反映更新后的视图 v'（由put 接收），也就是说，其视图（由get获取）与之相符 v'。可接受性属性（也称为 GetPut [14], ）指出，如果视图（由get获取）没有更新，那么源也不会发生更新（即不会由put导致更新）。这两种往返属性统称为良好行为性，满足这些属性的透镜称为良态。有时还会考虑一条称为 PutPut [14] 的附加定律，其直观含义是put 会保留更新的复合。我们忽略这条定律，因为一般认为它过于严格，会妨碍在基于状态的系统中进行有用的变换 [14, 26] ；在比较我们提出的框架与基于操作的系统时，我们将在第6节重新讨论这条定律。

可接受性蕴含了一个重要性质——稳定性，在文献中称为 PutGetPut [39]。

$$\forall s_0, s \in S, \forall v \in V.\ put\ \ell\ (s_0, v) = s \implies put\ \ell\ (s, get\ \ell\ s) = s \qquad \text{(stability)}$$

“One round-trip leads to a stable source and view” (P3)

当 $put\ \ell\ (s_0,\ v)$ 成功后，得到更新后的源 s，然后使用 $get\ \ell\ s$ 提取其视图时，所得的二元组（s，$get\ \ell\ s$）处于稳定状态：之后对它们进行任何进一步的get或put 都不会产生任何影响。换言之，一次往返——put再get——就足以使二者同步，从而涵盖整个更新传播过程。

3.2 定义域：具有相同更新的偏序集

如第 1 节和第 2 节所述，我们使用偏序集（简称偏序集）表示状态，这些状态称为部分指定状态，并具有由偏序 $\leq$定义的更新保持概念。我们还假设，我们的定义域配备了一组相对于 s 的相同更新I_s ，以便陈述与可接受性相对应的法则。

定义1. 域 $P = (S, \leq, I)$，称为*i*-偏序集，是一个三元组，其中 $(P, \leq)$ 是一个偏序集，且$I \subseteq S \times S$是 $(\leq)$ 的一个自反子集。

对于 $P = (S, \leq, I)$，我们将 $|P|$ 记作 S，并将I_s 记作 $\{s' \mid (s', s) \in I\}$。要求 $I \subseteq (\leq)$的直观含义是 $s \leq s'$，也就是说，s的具体程度低于 s'，也可以视为 s 相对于 s'不进行更新；从概念上说，I 是 $\leq$ 的一个子偏序，用于比较相同更新。我们不要求I 满足传递性，只是因为在形式化推导中不会用到这一性质。（I 的反对称性由$I \subseteq (\leq)$推出。）当一个部分指定状态表示的不只是一个可能的适当状态集合时，这两个关系之间的差异就很重要。当 P 具有最小元 Ω（或 P 有下界）时，我们还假设对任意 s，都有 $\Omega \in I_s$，这意味着，不指定任何内容的更新可以作为任意状态的相同更新。

i-偏序集的简单示例包括离散偏序集，其中 $(\leq) = (=)$，这也意味着 $I = (=)$；以及通过向离散偏序集添加一个不同的最小元而得到的偏序集，其中最小元 Ω 表示“任何事物”。当部分指定状态仅表示可能的完整状态集合时，由给定的完整状态集合 S 的幂集得到的 i-偏序集 $(2^S, (\supseteq), (\supseteq))$ 提供了最大的灵活性。可以将表示错误指定的 $\emptyset$排除在状态空间之外，转而采用 put 的失败。此时，并 $(\cap)$ 变为偏的。在这些简单示例中，I 与 $\leq$相同，但在下面这类非平凡示例中，两者的差异便显而易见。

例 *1*（界定相同更新）。通常，部分指定状态会对其原始（完整）状态作出某种条件假设。在这种情况下，$\leq$ 与 I 之间的差异就很重要。为说明这一点，考虑在 $K \rightharpoonup V$ 中表示为偏函数的键值映射，其中键集为 K，值集为 V。考虑表示映射删除请求的部分指定状态，它们自然可以表示为 K的子集 D。我们可以通过集合并轻松合并两个删除请求；删除 D 是否反映在 f 中，则形式化为 $D \cap \mathrm{dom}(f) = \emptyset$。

该域有一些可能的选择，可以同时容纳完整状态 f 和部分指定状态 D。一种选择是 $P_1 = ((K \rightharpoonup V) \uplus 2^K, (\leq), (\leq))$，其中 $(\leq)$ 是包含 (D, D') 且满足 $D \subseteq D'$ 以及包含 (D, f) 且满足 $D \cap \mathrm{dom}(f) = \emptyset$ 的最小自反关系。然而，由于 P_1 允许过多的“相同更新”，例如 K（直观上表示“删除所有内容”）对于 $f = \emptyset$而言，因此很难或很复杂地定义合法透镜，而合法透镜必须保留相同更新。一个更实际的选择是 $P_2 = ((K \rightharpoonup V) \uplus 2^K, (\leq), \mathrm{I})$，其中 I 是包含 (D, D') 且满足 $D \subseteq D'$ 以及包含 (D, f) 且满足 $D = \emptyset$ 的最小自反关系，从而排除在 I_f 中删除不存在的标识符。□

我们甚至可以通过将更新 [2, 11, 21, 38] 与其起始点配对，将其编码为部分指定状态。详情请参见第 5.1 节。

正如我们所见，域的种类多种多样，因此我们的形式化旨在独立于部分指定状态的任何具体表示：对这些域（源和视图状态的集合）的唯一假设是它们构成 i-偏序集。（相比之下，某些具体的基本透镜和组合子，例如第 2 节所示的那些，需要具体的 i-偏序集，或施加进一步的

假设，以确保其功能性和/或良好行为性。）人们或许会因此疑惑，为什么我们不像定义更新那样，定义它们在完整状态上的作用。我们确实这样做了，但没有将其纳入域的定义，而是通过称为部分状态初始化器的某些基本透镜（第 3.7 节）将其内化。

3.3 （无律）部分状态透镜

请注意，本文的主要目标是对处理部分指定数据的透镜给出形式化处理。既然我们已经将（可能是）部分指定数据形式化，现在就可以定义（无律版本的）部分状态透镜了。

定义 2（（无律）部分状态透镜）。对于 i-偏序集 P 和 Q，P 与 Q 之间的部分状态透镜（部分状态透镜）ℓ 是一对函数 $get\colon\ |P| \to |Q|$ 和 $put\colon\ |P| \times |Q| \rightharpoonup |P|$。□

我们将 Lens$^{\leq}\ P\ Q$ 写作 P 与 Q 之间的部分状态透镜 $\ell =$ （$get,\ put$）的集合，并将 $get\ \ell$ 和 $put\ \ell$ 分别写作它们的第一个和第二个组件。定义 2 表明，在没有律的情况下，部分状态透镜与经典透镜 [14]并无不同。在下文中，除非会引起混淆，我们有时会直接将部分状态透镜称为“透镜”；它们确实是无律透镜。

我们展示一些部分状态透镜的例子（它们确实满足定律）。

示例 *2*（恒等透镜）。我们最简单的透镜是恒等透镜 $idL \in$ 透镜$^{\leq}\ P\ P$，定义如下：

$$get\ idL\ s = s \qquad put\ idL\ (_, v) = v$$ □

例 *3*（常量透镜）。有时，视图数据的一部分不受源影响，例如 XML 视图中的元素名称。对于有下界（即具有最小元 Ω）且带有$a \in |Q|$ 的情况，常值透镜 $constL_a \in$透镜$^{\leq}\ P\ Q$ 是一个实用的构建模块。

$$get\ constL_a\ _ = a \qquad put\ constL_a\ (_, v) = \Omega \text{ if } v \in I_a$$

这里，$v \in I_a$ 而非 $v = a$ 是有意为之，也是透镜满足定律所必需的；直观地说，这确保了它与行为类似于常量透镜的透镜组合时也能正常工作。这个透镜有两个有趣之处。首先，它的 put 是刻意设计为部分函数（除非 $|Q| = \{a\}$）。其次，通过返回最小元 Ω，它明确表明源可以是任意值。□

下一个部分状态透镜更为复杂，并假设 i-偏序集具有额外结构，才能使其良态。

定义 3（可复制）。如果 i-偏序集 P具有一个指定的部分算子 $\oplus \in |P| \times |P| \rightharpoonup |P|$（称为 P 的合并算子），并且满足以下两个条件，我们就称它为可复制。

- $x \oplus y = z$ 蕴含 $x \vee y = z$，并且– 对于任意 x、y 和 z，若 $x \in I_z$ 且 $y \in I_z$，则 $x \oplus y \in I_z$ 成立。

□

也就是说，$\oplus$ 能够可靠地计算并，且对于固定状态的相同更新，它必须是全定义且封闭的（因此必须与 $\vee$一致）。合并算子必须满足幂等性；由于对任意 x，$x \in L_x$ 均成立，根据第二个条件，$x \oplus x$ 必须有定义；根据第一个条件，结果必须为 x 。可重复性的一个充分条件是 $\oplus$ 完备（（$\oplus$） =（$\vee$）），并且具有结合性，即 $x \oplus$（$y \oplus z$）与（$x \oplus y$）$\oplus z$ 在是否有定义及结果上均一致。对于简单的 i-偏序集，这一条件很容易满足，但对于定制的 i-偏序集，通常难以或不易验证。通过 i-偏序集的基本构造（如 $P \times Q$、$P + Q$ 和 P_Ω（通过添加一个新的最小元 $\Omega \notin |P|$，从 P 构造出的 i-偏序集）），可保持可重复性；此处假设对 $\leq$、I 和 $\oplus$采用逐点操作。

例*4.* 我们的复制透镜 $dup \in \mathsf{Lens}^{\leq} P (P \times P)$ 针对可复制的 P 定义如下：

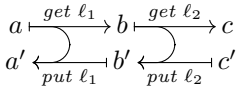
$$get\ dup\ s = (s, s) \qquad put\ dup\ \big(_, (s_1, s_2)\big) = s_1 \oplus s_2$$

此处，视图数据只是源数据的一个二元组。get 十分简单，它返回一个视图，其中包含原始源数据的两个相同副本。对于put，我们用两个（可能已更新的）副本的并来替换源；其直观含义是，通过合并部分指定状态，将它们的意图（例如第 2.3 节中提到的添加或删除特定任务）整合为一个统一的意图。例如，对于 $P = \{1,2\}_\Omega \times \{1,2\}_\Omega = \{(\Omega,\Omega),\ (1,\Omega),\ (\Omega,2),\ (1,2)\}$，采用由 $\Omega \leq 1,2$ 诱导的逐点序，这意味着 Ω 对应分量上的“空操作”；并采用合并算子 $(\oplus) = (\vee)$，其中 Ω 是单位元，则有 $put\, dup\ \big(s,\ (1,\Omega),\ (\Omega,\ 2)\big) = (1,\ \Omega) \oplus (\Omega,\ 2) = (1,\ 2)$。

□

最后，我们还可以按照如下标准方式定义两个部分状态透镜$\ell_1 :\ \mathsf{Lens}^{\leq}\ P\ Q$ 和 $\ell_2 :\ \mathsf{Lens}^{\leq}\ Q\ R$ 的复合 $\ell_1 \fatsemi \ell_2$：

$$\begin{aligned} &get\ (\ell_1 \fatsemi \ell_2)\ a = get\ \ell_2\ (get\ \ell_1\ a) \\ &put\ (\ell_1 \fatsemi \ell_2)\ (a, c') = put\ \ell_1\ \big(a, put\ \ell_2\ (get\ \ell_1\ a, c')\big) \end{aligned}$$



复合具有结合性，并且在示例 2 中具有恒等透镜左恒等元[8]。

3.4 定律：部分状态一致性和可接受性

现在，我们聚焦于部分状态透镜的定律。首先，我们将经典透镜的一致性和可接受性 [4,14] 调整到部分状态透镜的情形，同时保留其直接、非正式的意图（P1 和 P2）。稳定性（P3）暂且搁置

[8] 当 get 可以是部分状态时，要使 idL 成为右恒等元，还需要满足以下条件：$put\ \ell\ (s,\ v) = s'$ 蕴含 $s \in \mathsf{dom}(get\ \ell)$。换言之，$put$ 仅对原始源 s 有定义，而 get 也正是对该源有定义的。这不会造成任何实际限制，因为调用 put 假定 get的成功执行。

稍后再讨论；正如我们很快将看到的，在我们的情形下，P1 和 P2 的扩展并不能保证 P3，因此我们还需要一个额外性质。由于定义域是偏序的，自然会要求 get 和 put 变换具有单调性。我们对一致性和可接受性的定义共同保证了 get 的单调性（引理 2），但我们有意允许 put 不是单调的。

部分状态一致性 我们首先关注一致性（P1）：

$$\forall s, s' \in S, \forall v' \in V.\ put\ \ell\ (s, v') = s' \implies get\ \ell\ s' = v' \qquad \text{(一致性)}$$
"Updates to the source's view are preserved in the updated source." (P1)

因为它与透镜 dup 直接相关，而该透镜在我们的动机示例中发挥着核心作用。

回想一下，dup 引入了源共享，这会导致副本上的更新合并，从而违反一致性。例如，对于任意 s，我们有 $put\ dup\ (s,\ ((1,\ \Omega),\ (\Omega,\ 2))) = (1,\ 2)$，而更新后的源的视图 $((1,\ 2),\ (1,\ 2))$ 不同于任一更新后的视图 $(1,\ \Omega)$ 和 $(\Omega,\ 2)$。如第 2.3 节所述，我们用 $\leq$ 表示更新保持：$put\ \ell\ (s,\ v') = s'$ 必须蕴含 $v' \leq get\ \ell\ s'$，这意味着更新后的视图中的更新意图 v' 在更新后的源的视图 $get\ \ell\ s'$ 中得到保持。为了使该性质可组合，我们确保对于任意满足 $s' \leq s''$ 的 s''，都有 $v' \leq get\ \ell\ s''$，从而确保 ℓ 可以与同类透镜进行前置复合。总之，$\ell \in \mathsf{Lens}^{\leq}\ P\ Q$ 的部分状态一致性形式化如下，并满足 P1。

$$\forall s, s' \in |P|, \forall v' \in |Q|.\ put\ \ell\ (s, v') \leq s' \implies v' \leq get\ \ell\ s' \qquad \text{(ps-consistency)}$$

部分状态可接受性接下来，我们调整可接受性条件（P2）：

$$\forall s \in S, \forall v \in V.\ get\ \ell\ s = v \implies put\ \ell\ (s, v) = s \qquad \text{(可接受性)}$$
"No updates to the source's view means no updates to the source." (P2)

诚然，原始版本具有可组合性，并能如所要求地保证稳定性，但由于以下两个原因，它对我们的目的而言过于强硬。

– 我们希望 put 返回尽可能小的结果，从而使 $\vee$ 进一步进行合并时更有可能成功。（对于一般的偏序集，这一点并不成立，但通常适用于我们的定义域。）一个极端例子是例 3 中的 $constL_{42}$，其中 $put\ constL_{42}\ (s_0, 42) = \Omega$，这表示该透镜完全不包含更新后的源，并且与 Ω 的任何合并总能以 $\Omega \vee s = s \vee \Omega = s$ 成功。– 原始定律妨碍了透镜作为更新转换器的自然行为。回想一下，第 2.3 节中过滤透镜 $filter_{\neg \mathrm{Done}}$ 的 put 会将视图上的插入请求（即带有（+）标记的元组）原封不动地传递到源中。这种看似无害的行为违反了原始定律，因为原始源可能包含无法在视图上表示的更新请求，例如已完成任务的插入请求。其 get 或 put 必须筛选掉或拒绝此类更新，才能实现更新保持。

因此，我们针对这些行为调整了原始的可接受性定律。根据非正式要求 P2（即 no-update 或相同更新会被 put 保持），我们给出如下部分状态可接受性的定义。

$$\forall s \in |P|, \forall v \in |Q|.\ v \in I_{get\ \ell\ s} \implies put\ \ell\ (s, v) \in I_s \qquad \text{(ps-acceptability)}$$

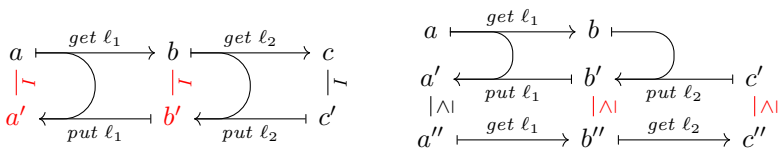
原始版本对应于 I 为相等关系（$=$）的情形，或者等价地，对于两个定义域上的任意 s，都有 $I_s = \{s\}$。

部分状态弱良定义性 到目前为止，我们已经获得了一致性和可接受性的部分状态版本，这两者是该文献中使用的标准往返属性 [4,14,17]。我们将这两个属性统称为弱良定义性。之所以称其为“弱”，是因为它不涵盖稳定性；我们将在第 3.5 节通过添加一条定律来恢复稳定性。尽管它是“弱”的，但当（$\leq$）为（$=$）时，由于 I 作为（$\leq$）的自反子集，也意味着它同样是（$=$），弱良定义性便与原始的良好行为性相一致。从这个意义上说，弱良定义性是对原始概念的自然推广。

我们陈述部分状态弱良定义透镜的一些性质。首先，它们的两个定义性质在透镜复合下是封闭的：

引理 1. *ps*-可接受性和 *ps*-一致性在透镜复合下均封闭。

证明（概要）。见下图；黑色部分是已知的，红色部分将予以证明。



为证明 $\ell_1 \fatsemi \ell_2$ 的 ps-可接受性，我们首先利用 ℓ_2 的 ps-可接受性证明 $b' \in I_b$，然后利用 ℓ_1 的 ps-可接受性得出 $a' \in I_a$。为证明 $\ell_1 \fatsemi \ell_2$ 的 ps-一致性，我们首先利用 ℓ_1 的 ps-一致性，然后利用 ℓ_2 的 ps-一致性依次证明 $b' \leq b''$ 和 $c' \leq c''$。

其次，要使 ℓ 弱良构，$get\ \ell$ 必须是单调的。

引理 2. 对于弱良构的部分状态透镜 ℓ，若 $get\ \ell\ s = v$ 和 $s \leq s'$，则有 $v \leq get\ \ell\ s'$。□

推论1。 对于弱良构的部分状态透镜 ℓ，$get\ \ell$ 是单调的。 □

相比之下，我们的 put 在第二个参数上不一定是单调的。通常，部分状态初始化器会违反该条件，因为它们往往会尽可能保留源的原始信息；因此，当它们接收一个更小的部分指定状态时，更新后的源中会保留更多源的原始信息。例如，对于第 2 节中的 $init_{\text{tasks}}$，注意到 $put\ init_{\text{tasks}}\ (s_{\text{tl}}, \Omega) = s_{\text{tl}}$

而 $put\ init_{\mathrm{tasks}}\ (s_{\mathrm{tl}}, w_{\mathrm{merged}}) = s''_{\mathrm{tl}}$，其中 $s_{\mathrm{tl}} \not\leq s''_{\mathrm{tl}}$。还有一种合法透镜，其 put 在第一个参数上不是单调的（见附录 B）；与第二个参数的情况不同，我们没有发现任何实际理由需要这种透镜——之所以允许它们，只是因为我们的要求（定律）是最低限度的。

第三，反复进行往返操作不会改变视图。这在文献中称为 GetPutGet [39], ，在经典情形下可由可接受性(GetPut)推出。

引理 3（视图稳定性）。对于弱良构部分状态透镜 ℓ，任意满足 $get\ \ell\ s = v$ 的 s，都有 $get\ \ell\ (put\ \ell\ (s, v)) = v$。□

第四，与经典透镜类似， [12], put 可以唯一确定 get，前提是满足弱良定义性。

引理 4。设 $\ell \in \mathsf{Lens}^{\leq}\ P\ Q$ 是一个弱良构的部分状态透镜，且V_s是集合 $\{v \in |Q| \mid \exists s_0, s_1 \in |P|.\ put\ \ell\ (s_0, v) = s_1 \wedge s_1 \leq s\}$。那么，$get\ \ell\ s = \max V_s$。

引理 4 为在给定相应的 put 时定义部分指定状态的 get 提供了指导；get 实际上由 put 决定，我们别无选择。计算 V_s 往往并不简单，因为可能发生超调（$s_1 \leq s$），但该引理有助于筛选候选项；例如，如果 put 对不同的 v_1 和 v_2（满足 $v_1 \leq v_2$）都返回相同的 s，那么它的 get 就不能返回 v_1，因为 v_1 不可能是 $\max V_s$。与 put 不同，get 并不总能确定 put，这也与经典透镜类似。我们很快会看到，良态经典透镜也是良态（因而也是弱良构）的部分状态透镜（引理 7）。

3.5 附加定律：部分状态稳定性

现在，我们有了 ps-可接受性和 ps-一致性定律，它们分别满足性质 P2 和 P1。接下来的目标是找到一条调整稳定性的定律，使其满足 P3，同时支持组合推理。

$$\forall s_0, s \in S, \forall v \in V.\ put\ \ell\ (s_0, v) = s \implies put\ \ell\ (s, get\ \ell\ s) = s \qquad \text{(stability)}$$
$$\textit{“One round-trip leads to a stable source and view”} \qquad \text{(P3)}$$

这条稳定性定律很重要。它指出，一次往返操作——用 put 更新源，然后用 get 获取其视图——就能涵盖达到稳定状态所需的整个更新过程（s，$get\ \ell\ s$）：之后再对该对执行任何 get 或 put 操作都不会产生影响。

问题：非稳定性 如前所述，仅有弱良定义性并不能保证稳定性。可接受性属性蕴含稳定性，但在我们的情形中，我们只有较弱形式的可接受性，因此稳定性成了一个问题。部分状态透镜面临的挑战在于，put 可能会产生一个部分指定的状态，例如透镜 $constL_{42}$。借助 ps-可接受性，我们有$put\ \ell\ (s, get\ \ell\ s) \in I_s$，这意味着 $put\ \ell\ (s, get\ \ell\ s) \leq s$，但也可能出现这种情况

即 $put\ \ell\ (s,get\ \ell\ s) < s$。尽管 $constL_{42}$ 恰好是稳定的，因为稳定性关注的是原始源是通过 put 得出的情况（因此，$s = \Omega$），但这种底层行为——put 可能返回一个比原始源更小的状态——可以被利用来构造不稳定的透镜，如下所示。

例 5（不稳定透镜）。令 $\mathbf{1}$ 为单元素 i-偏序集，其唯一元素为（），令 N 为集合 $\{0,1,2,\dots\}$，其中 $i \leq j$ 当且仅当 $i \leq j$，并且 $I = (\leq)$。[9] 直观地说，元素 $n \in N$ 表示更新优先级（不带负载）。考虑透镜 bad：$\mathsf{Lens}^{\leq}\ N\ \mathbf{1}$ 定义如下：

$$get\ bad\ _ = ()\qquad put\ bad\ (s,()) = \begin{cases} \underline{0} & \text{if } s = \underline{0} \\ \underline{k} & \text{if } s = \underline{k+1} \end{cases}$$

由于 $put\,bad(n,\ _())$ 在 n 非零时会产生 $n-1$，因此需要将 $put\,bad(-,\ ())$ 应用 n 次，n 才能达到不动点 0，未能满足 P3。

无论如何，透镜 bad 仍同时满足 ps-可接受性和 ps-一致性；前者归结为 $put\,bad(s,\ ()) \in I_s$，这是显然的；后者则是平凡的，因为唯一可能的视图是单位值（）。因此，弱良定义性不足以排除这一不良透镜。□

备注。如果最终源是离散的——即只包含具体状态，这通常是理想的——那么稳定性自然成立。这是因为在离散域中 $(\leq) = I = (=)$，因此 ps-可接受性保证了 $put\ \ell\ (s',get\ \ell\ s') = s'$。然而，不稳定透镜的 put 未必总能给出“确定的答案”，例 5 展示了这一点：执行多次往返操作可以逐步细化源。因此，对于不稳定透镜的复合，虽然整体 put 可能失败，但额外的局部往返操作或许能成功传播更新。我们还注意到，正如第 2.5 节所述，当透镜需要借助无冲突复制数据类型 [3, 44] 来接受多个视图上的任意同步更新时，最终源中包含严格部分指定状态是有用的。

部分状态稳定性 因此，我们需要为 P3 增加一项性质。直接强制满足稳定性定律的问题在于，它与组合推理相冲突，而组合推理是透镜作为双向变换方法所依据的核心原则。众所周知，稳定性本身在复合下并不封闭（参见 Matsuda 等人 [33, 附录 B]）。

由于 ps-可接受性已经保证 $put\ \ell\ (s',get\ \ell\ s') \in I_{s'}$，这意味着 $put\ \ell\,(s',get\ \ell\ s') \leq s'$，因此只需强制要求 $s' \leq put\ \ell\,(s',get\ \ell\ s')$ 成立，即可确保稳定性。为以可组合的方式强制这一点，假设满足 ps-一致性和

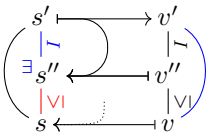
[9] 因此，N 是按标准序排列的自然数集合。此处我们不使用 $\mathbb{N}$，因为在通常意义下，$\mathbb{N}$ 中的 1 和 2 在规定性方面是不相容的；它们都是适当状态。

ps-可接受性，我们要求满足以下定律。

$$\forall s_0, s, s', s'' \in |P|. \forall v, v'' \in |Q|.\\ \big((put\ \ell\ (s_0, v) = s \leq s') \ \wedge\ (v \leq v'' \in I_{get\ \ell\ s'}) \ \wedge\ (put\ \ell\ (s', v'') = s'') \big)\\ \Longrightarrow\ s \leq s''$$

（ps-稳定性）

借助右侧的图示，或许更容易理解这条定律：图中，推导出的部分用红色表示，而由 ps-可接受性和 ps-一致性推导出的部分用蓝色表示。这里，我们用 s 表示$put\ \ell\ (s_0,\ v)$，用 v' 表示$get\ \ell\ s'$。我们将较大的元素置于



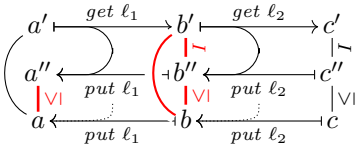
上方的行中，以突出元素之间的序。（由于 s_0 与序无关，因此省略，并在图中用虚线表示省略的输入。）

这条定律旨在使其在复合下保持成立。条件 $s \leq s'$和 $v'' \in I_{get\ \ell\ s'}$（而非 $s = s'$ 和 $v'' = get\ \ell\ s'$）体现了 ps-一致性和 ps-可接受性的设计思路。相较之下，由复合强制满足的条件 $v \leq v''$对于使这条定律具有实用性至关重要；否则，示例 2 中的左恒等元 $\in$透镜$^{\leq}$ P P对一般的 P并不满足该性质（例如，取 $s_0 = \Omega$、 $v = 1$、 $s' = s = 1$和 $v'' = \Omega$ 即可得到 $s'' = \Omega$）。$put\ \ell\ (s',\ v'')$有定义这一点被列为定律的要求，而非结论；不过这并无影响，因为 ps-可接受性已蕴含了这一点。

形式化地说，我们有以下引理5和引理6。

引理5。 对于满足 *ps*-可接受性、*ps*-一致性和 *ps*-稳定性的 ℓ_1 和 ℓ_2 ， $\ell_1 \fatsemi \ell_2$ 也满足 *ps*-稳定性。

证明。 考虑右侧的图。
该图展示了$put\ (\ell_1 \fatsemi \ell_2)\ (a_0, c)\ = a$、$get\ (\ell_1 \fatsemi \ell_2)$ $a' = c'$和$put\ (\ell_1 \fatsemi \ell_2)\ (a', c'')\ = a''$的执行过程，其中包含 $a \leq a'$和 $c \leq c'' \leq c'$。这里，我们用粗红色箭头标出需要证明的部分。



根据 ℓ_1 和 $a \leq a'$的 ps-一致性，我们有 $b \leq b'$。接着，我们利用 ℓ_2 的 ps-稳定性得到 $b \leq b''$。根据 ℓ_2 和 $c'' \leq c'$的 ps-可接受性，我们还有 $b'' \in I_{b'}$。于是，根据 ℓ_1的 ps-稳定性，我们得到 $a \leq a''$，证毕。□

引理6。 如果一个部分状态透镜 ℓ 满足 *ps*-可接受性、*ps*-一致性和 *ps*-稳定性，那么它也满足稳定性。

证明。假设对于某些 s_0 和 v，有 s = put ℓ (s_0,v)。接下来，我们证明 put ℓ (s,get ℓ s) 被定义为 s。然后，ps-一致性确保 $v \leq$ get ℓ s，而 ps-可接受性确保 s'' = put ℓ (s, get ℓ s) 有定义，并且满足 $s'' \in$I$_s$。现在，我们可以将 ps- 稳定性应用于 $s' = s$ 和 v'' = get ℓ s，从而得到 $s \leq s''$。由于根据 $s'' \in$ I$_s$ 有 $s'' \leq s$，由 $\leq$的反对称性可得 $s = s''$ = put ℓ (s, get ℓ s)。□

现在，我们可以定义部分状态透镜的良好行为性。

定义 4（部分状态透镜的良好行为性）。部分状态透镜 ℓ 是良态的，当且仅当它是弱良构的且满足 ps-稳定性。□

换句话说，一个良态的部分状态透镜满足 ps-可接受性、ps-一致性和 ps-稳定性。这些性质分别描述了透镜在 get-后接-put、put-后接-get，以及 put-后接-get-后接-put 情形下的行为。

定理 1. 良好行为性在复合下封闭。也就是说，当 ℓ_2 和 ℓ_1 良态时，$\ell_1 ; \ell_2$ 也良态。□

当 P of $\ell \in \text{Lens}^{\leq}\ P\ Q$ 是离散的（即 $(\leq) = I\ = (=)$）时，ps-稳定性可由 ps-可接受性推出；在这种情况下，良态与弱良态一致。此外，当 Q 也离散时，我们的（弱）良态性与经典良定性一致，这表明我们的形式化是经典透镜框架的简洁扩展。

引理 7。对于 $\ell \in$ 透镜$^{\leq}\ P\ Q$，若 P、ℓ 为离散偏序集，则它是弱良构的，当且仅当它是良态的。□

引理 8。经典透镜 $\ell \in$ 透镜 $A\ B$ 是良态的，当且仅当 $\ell \in$透镜$^{\leq}\ (A, =, =)\ (B, =, =)$ 作为部分状态透镜是良态的。□

3.6 良态部分状态透镜示例

接下来，我们给出一些良态部分状态透镜的示例。

示例 *6*（恒等透镜）。示例 2 中的恒等透镜 左恒等元 $\in$ 透镜$^{\leq}\ P\ P$ 对于任意 i-偏序集 P都是良态的。□

示例 *7*（常值透镜）。示例 3 中定义在有下界的 P 上的常值透镜 $constL_a \in$透镜$^{\leq}\ P\ Q$ 是良态的。其 ps-可接受性显然，因为它的 put 对任意 $v \in I_a$ 都有定义，并返回最小元。此外，其 ps-一致性也显然，因为无论输入是什么，它的 get 总是返回 a。由于对于任意 s_0 和 v，$put\ \ell\ (s_0, v) = \Omega$，ps- 稳定性也显然：无论 s'' 如何，我们都有 $s \leq s''$。□

示例 *8*（复制透镜）。例4中的复制透镜 $\delta \in \text{Lens}^{\leq}\ P(P \times P)$ 在 P 可复制时是良态的。

为确保 ps-可接受性，只需检查对于任意满足 $v_i \in \text{I}_s\ (i = 1, 2)$ 的 (v_1, v_2)，是否有 put dup $(s, (v_1, v_2))\ = (v_1 \oplus v_2) \in \text{I}_s$。由于 P 可复制，$v_1 \oplus v_2$ 有定义，且满足 $(v_1 \oplus v_2) \in \text{I}_s$。为确保 ps-一致性，由于 get 单调，只需检查当 put dup $(s_0, (v_1, v_2)) = (v_1 \oplus v_2, v_1 \oplus v_2)$ 有定义时，是否有 $(v_1, v_2) \leq$ get dup $s = (s, s)$（因此可以将 $\oplus$ 替换为 $\vee$）。这由 $\vee$ 的定义直接推出。为确保 ps-稳定性，只需证明当 $\leq$ 两侧都有定义时，$(v_1 \oplus v_2) \leq (v_1' \oplus v_2')$ 满足 $v_i \leq v_i'\ (i = 1, 2)$（因此可以将 $\oplus$ 替换为 $\vee$）。由于对于 $i = 1, 2$，$v_i \leq v_i' \leq (v_1' \vee v_2')$ 成立，且 $v_1 \vee v_2$ 是 v_1 和 v_2 的最小上界，因此有 $(v_1 \vee v_2) \leq (v_1' \vee v_2')$。□

反例 *9.* 例 5 中不理想的透镜 bad 不是良态的，因为它违反了 ps-稳定性。取满足 $s = put\, bad(s_0, ())\ = 1$ 的 $s_0 = 2$，再取 $s' = s$，则有 $put\, bad(s', ())\ = put\, bad(1, ())\ = 0 \not\geq 1 = s = s'$，这违反了该性质。□

例 *10*（去标签）。到目前为止，我们见过的许多基本透镜都没有在put中使用其第一个参数。当然，情况并非总是如此。下面这个透镜$unTagS\ \in \mathsf{Lens}^{\leq}\ (P + P)\ P$就是一个这样的例子。

$$\begin{array}{ll} get\ unTagS\ (\mathsf{InL}\ s_1) = s_1 & put\ unTagS\ (\mathsf{InL}\ _)\ s = \mathsf{InL}\ s \\ get\ unTagS\ (\mathsf{InR}\ s_2) = s_2 & put\ unTagS\ (\mathsf{InR}\ _)\ s = \mathsf{InR}\ s \end{array}$$

请注意，它的put 只利用源信息来确定标签；这就是该透镜被称为$unTagS$的原因。

我们还有一个一般形式$unTag\ \ \phi_1\ \phi_2 \in \mathsf{Lens}^{\leq}\ (P_{\phi_1} + P_{\phi_2})\ P$，它使用源和视图信息以及单调谓词$\phi_1$ 和 ϕ_2来确定标签。

$$put\ (unTag\ \phi_1\ \phi_2)\ s\ v = \begin{cases} \mathsf{InL}\ v & \text{if } (s = \mathsf{InL}\ _ \wedge \phi_1\ v) \vee (s = \mathsf{InR}\ _ \wedge \phi_1 v \wedge \neg(\phi_2\ v)) \\ \mathsf{InR}\ v & \text{if } (s = \mathsf{InR}\ _ \wedge \phi_2\ v) \vee (s = \mathsf{InL}\ _ \wedge \phi_2 v \wedge \neg(\phi_1\ v)) \end{cases}$$

这里，P_ϕ 表示其元素为 $\{x \in |P| \mid \phi\ x\}$的 i-偏序集。对于任意 i-偏序集 P，它们都是良态的。

透镜$unTagS$/$unTag$ 可作为条件分支的构建模块。由于篇幅所限，我们省略具体定义，基本思路是：使用透镜 $\mathsf{Lens}^{\leq}\ S\ (A + B)$ 检查条件，通过透镜 $\mathsf{Lens}^{\leq}\ A\ C$ 和 $\mathsf{Lens}^{\leq}\ B\ C$ 分别处理每种情况以得到 $C + C$，最后再用 $unTagS$/$unTag$ 擦除标签（对于$unTag$，C 可以由不同的谓词针对各个分支加以细化）。这一行为对应于经典透镜 [14]中的$ccond$/$cond$ 。□

3.7 偏序与更新的关联：部分状态初始化器

我们给出 ps-发起器的形式化定义。尽管在我们的框架中，它们与其他透镜并无不同，但其一般形式阐明了部分指定状态、完整状态、更新与偏序之间的关系。

正如第 2.3 节中非正式提到的，ps-发起器的定义 初始化 $\in$透镜$^{\leq}\ S\ P$，适用于离散 S 的 $S \subseteq P$，遵循相同的模式。

$$get\ init\ s = s \qquad\qquad put\ init\ (s, v) = v \cdot s$$

也就是说，其 get 在（可能的）部分指定状态P中嵌入完整状态 S，而其 put 则通过 $(\cdot) \in P \to S \rightharpoonup S$，将部分指定状态的语义实现为更新。由于 S 是离散的，若要使 $init$ 满足良态，只需满足 ps-可接受性和 ps-一致性；它们可归结为以下两个性质：

$$\begin{array}{ll} \forall v \in |P|, s \in |S|.\ v \in I_s \Longrightarrow v \cdot s = s & \text{(u-acceptability)} \\ \forall v \in |P|, s, s' \in |S|.\ v \cdot s = s' \Longrightarrow v \leq s' & \text{(u-consistency)} \end{array}$$

直观地说，$v \in I_s$ 表示 v 相对于 s不代表任何更新——这正是 u-可接受性所确保的。u-一致性属性指出，将更新 v 应用于状态 s 的应用程序 $v \cdot s$ 若成功，就必须保留以 v给出的意图。

4 重新审视场景：形式化实现

现在，我们可以重新审视第 2 节中的示例，并将先前非正式给出的解决方案形式化。我们首先定义与第 2.3 节示例相对应的版本（第 4.1–4.3 节），然后针对第 2.4 节中介绍的更细粒度更新对其进行扩展（第4.5节）。

在定义转换中涉及的具体定义域（任务、DT）之前，我们先给出其整体结构，如下所示。

$$\ell_{\text{task}} \in \mathsf{Lens}^{\leq}\ \mathsf{Tasks}\ (\mathsf{DT} \times \mathsf{DT})$$
$$\ell_{\text{task}} = \mathit{init}_{\text{tasks}} \fatsemi \mathit{dup} \fatsemi (\mathit{filter}_{\neg \text{Done}} \times \mathit{filter}_{\text{Apr1}})$$

上述内容涉及以下透镜和透镜组合子（以及 ⨟）。

$\mathit{dup} \in$ 透镜$^{\leq}\ P (P \times P)\ (\times) \in$ 透镜$^{\leq}\ P_1\ Q_1 \to$ 透镜$^{\leq}\ P_2\ Q_2 \to$ 透镜$^{\leq}\ (P_1 \times P_2)\ (Q_1 \times Q_2)$
$\mathit{init}_{\text{tasks}} \in$透镜$^{\leq}$ 任务 DT $\mathit{filter}_{\neg \text{Done}} \in$透镜$^{\leq}$ DTDT $\mathit{filter}_{\text{Apr1}} \in$透镜$^{\leq}$ DTDT

这里，P 是任意可复制的 i-偏序集，而 P_1、P_2、Q_1 和 Q_2 是任意 i-偏序集。其中，我们已在第 3 节中见过duplication 透镜的定义，并确认了其良好行为性。接下来，在定义相关定义域之后，我们将给出$\mathit{init}_{\text{tasks}}$、筛选$_{\neg \text{Done}}$、筛选$_{\text{Apr1}}$ 和 $(\times)$ 的具体定义，并确认这些透镜（$\mathit{init}_{\text{tasks}}$ 和筛选器）是良态的，且透镜组合子 $(\times)$ 保持良好行为性。

4.1 定义域

粗略地说，Tasks 是由完整状态构成的离散偏序集，而 DT 还额外允许添加和删除更新。为简单起见，我们使用这些名称同时表示 i-偏序集及其底层集合。它们的定义与例 1 类似，但现在我们同时考虑添加和删除。我们将$t \in$ Tasks 视为一个偏函数，即 Tasks $\triangleq$ ID $\rightharpoonup$ (布尔值 $\times$ 字符串 $\times$日期)。严格部分指定状态的形式为 $(A,\ D)$，其中 $A \in$ Tasks 且 $D \subseteq$ ID，并满足 Disj$(D,\ \text{dom}(A))$ 的限制。这里，我们用 Disj$(S,\ T)$ 表示S与 T不相交，即 $S \cap T = \emptyset$。直观地说，$(A,\ D)$ 约束 A必须存在，而 D 不得存在；Disj$(D,\ \text{dom}(A))$ 确保可行性。形式上，我们定义 DT $\triangleq$ Tasks $\cup \{(A,\ D) \in \text{Tasks} \times 2^{\text{ID}} \mid \text{Disj}(D,\ \text{dom}(A))\}$，并给出如下序和相同更新（显然的自反性规则省略）。

$$\frac{A \subseteq A' \quad D \subseteq D'}{(A, D) \leq (A', D')} \qquad \frac{A \subseteq t \quad \text{Disj}(D, \text{dom}(t))}{(A, D) \leq t} \qquad \frac{A \subseteq A' \quad D \subseteq D'}{(A, D) \in I_{(A', D')}} \qquad \frac{A \subseteq t}{(A, \emptyset) \in I_t}$$

我们定义其合并算子 $\oplus$如下。

$$\begin{aligned} t \oplus t &= t \\ t \oplus (A, D) = (A, D) \oplus t &= t && \text{if } (A, D) \leq t \\ (A, D) \oplus (A', D') &= (A \cup A', D \cup D') && \text{if } (A \cup A') \in \mathsf{Tasks} \\ & && \quad \wedge \mathrm{Disj}(D \cup D', \mathrm{dom}(A \cup A')) \end{aligned}$$

最后一行的条件表示 $(A \cup A',\ D \cup D')$ 是 DT 中一个有效的部分指定状态。这 $\oplus$ 合理地实现了 DT 中的并，并且对于任意 x，在 I_x 上都是全定义且封闭的，这意味着 DT 是可复制的。实际上，这里$\leq$、I和 $\oplus$ 的构造是通用方案（第 5.1 节）的一个实例，而可重复性的条件很容易满足（引理 9）。这种 i-偏序集设计受到 2P-Set CRDT [44], 的启发，后者使用一对 $(A,\ D)$ 来表示集合 $A \setminus D$。

DT 元素的一个例子是第 2.3 节中的 w_{merged}，其中我们在表格中使用 marks(+) 或 (–)，而不是使用对 $(A,\ D)$。部分指定状态 w_{og} 和 w_{dt} 也是 DT 中的元素。

4.2 Ps-发起器：初始化$_{\mathbf{tasks}}$

ps-发起器 初始化$_{\mathrm{tasks}}$ 连接任务中的完整状态与 DT 中的部分指定状态。我们将其作为第 3.7 节所讨论的一般形式的一个实例给出。由于我们已经定义了其源域和视图域（任务和 DT），剩下的任务就是适当地定义 $(\cdot)$，使其满足 u-可接受性和 u-一致性。这是直接的:

$$t' \cdot _ = t' \qquad\qquad (A, D) \cdot t = \{(k \mapsto v) \in (t \lhd A) \mid k \notin D\}$$

这里，$t \lhd A$ 将 A 插入或更新到 t中；也就是说，对于每个 $k \in \mathrm{dom}(A)$，根据 $t(k)$ 是否有定义，它会使用 $A(k)$ 更新 $t(k)$，或将 $A(k)$ 插入t 中。形式化地，$(t \lhd A)(k)$ 定义为：如果 $k \in \mathrm{dom}(A)$，则为 $A(k)$；否则为 $t(k)$。例如，我们有$w_{\mathrm{og}} \cdot s_{\mathrm{tl}} = s'_{\mathrm{tl}}$ 和 $w_{\mathrm{merged}} \cdot s_{\mathrm{tl}} = s''_{\mathrm{tl}}$。

上面的插入或更新与删除的顺序并不重要，因为我们要求 $\mathrm{Disj}(D,\ \mathrm{dom}(A))$。提出这一要求是一种设计选择：没有这一要求时，可以合并更多部分指定状态，但更新保留的概念会被削弱。没有这一要求时，我们需要为 A 或 D规定优先级，这会使非优先项成为软约束（因为它可能会被覆盖）。

4.3 过滤透镜

接下来，我们给出筛选$_{\neg\mathrm{Done}} \in$ 透镜$^{\leq}$ DT DT 的定义。类似地，我们省略筛选$_{\mathrm{Apr1}}$ 的定义。在前向方向上，筛选$_{\neg\mathrm{Done}}$ 透镜只筛选进行中的任务。

$$\mathit{get}\ \mathit{filter}_{\neg\mathrm{Done}}\ s = \begin{cases} t_{\mathrm{OG}} & \text{if } s = t \in \mathsf{Tasks} \\ (A_{\mathrm{OG}}, D) & \text{if } s = (A, D) \end{cases}$$

这里，给定 $t \in$ 任务，我们用 t_{OG} 表示通过收集 t 中所有进行中的任务得到的映射，即 $t_{\mathrm{OG}} = \{(k \mapsto v) \in t \mid v = (\text{假}, _, _)\}$。在后向方向上,–如果更新后的视图是完整状态，它就必须返回一个完整状态。否则，连续的反向执行可能会添加更多任务，从而违反更新保持。具体来说，它会将 t' 插入或更新到原先被筛选掉的任务中。如果更新后的视图是部分指定状态，它就会原样返回该部分指定状态，而不考虑源 s。

$$put\ filter_{\neg \mathrm{Done}}\ (s, v) = \begin{cases} (t \setminus t_{\mathrm{OG}}) \lhd t' & \text{if } v = t' \in \mathsf{Tasks}_{\mathrm{OG}} \wedge s = t \in \mathsf{Tasks} \\ (A, D) & \text{if } v = (A, D) \wedge A \in \mathsf{Tasks}_{\mathrm{OG}} \end{cases}$$

我们将仅由进行中的任务组成的一组部分指定状态记作 $\mathsf{Tasks}_{\mathrm{OG}}$，即 $\{t \in \mathsf{Tasks} \mid \forall k,\ t(k) = (\text{假}, _, _)\}$。如果这是一个严格部分指定状态，此 put 会原样返回更新后的视图。这种行为背后的直觉是，严格部分指定状态表示更新请求，而对过滤结果的添加/删除请求可以转换为对原始任务的相应请求。

无论 D 是否属于进行中的任务，此透镜都会将其原样传递。这是安全的，因为该透镜是良态的（证明见附录 A.4）。对于 ps-一致性，只有在证明 $(A, D) \leq$get 过滤器 $\neg$已完成 $t = t_{\mathrm{OG}}$ 来自 put 过滤器$_{\neg \mathrm{Done}}(s, (A, D)) = (A, D) \leq t$时，这一点才有影响。即使$D$ 包含针对已完成任务的删除请求，我们仍有 $(A, D) \leq t_{\mathrm{OG}}$（前提是 $A \in$Tasks$_{\mathrm{OG}}$，而 put 的守卫条件保证了这一点）。注意，根据 $t_{\mathrm{OG}} \subseteq t$，Disj($D$, dom($t$)) 蕴含 Disj($D$, dom($t_{\mathrm{OG}}$))。回想一下，$D$ 约束带有这些标识符的任务不会出现；也允许有更多任务不出现（只要包含 A）。对于 ps-可接受性，无需担心这一点，因为 $(A, D) \in \mathrm{I}_t$ 只是强制要求 D 为空集。

4.4 透镜组合子 $\times$

$\times$ 的定义与文献 [41]中的定义并无不同。设 $\ell_1 \in$Lens$^{\leq}$ P_1 Q_1 和 $\ell_2 \in$ Lens$^{\leq}$ P_2 Q_2 为透镜。接着，我们定义透镜 $(\ell_1 \times \ell_2) \in$ Lens$^{\leq}$ $(P_1 \times P_2)$ $(Q_1 \times Q_2)$，它将 ℓ_1/ℓ_2 并行应用于该对的每个分量，如下所示。

$$\begin{aligned} &get\ (\ell_1 \times \ell_2)\ (s_1, s_2) = (get\ \ell_1\ s_1, get\ \ell_2\ s_2) \\ &put\ (\ell_1 \times \ell_2)\ ((s_1, s_2), (v_1, v_2)) = (put\ \ell_1\ (s_1, v_1), put\ \ell_2\ (s_2, v_2)) \end{aligned}$$

得益于逐点序，组合子 $\times$ 保持良好行为性。

4.5 用于更细粒度更新的扩展定义域

第 2.4 节提到，我们可以支持更细粒度的更新描述（严格部分指定状态）u_1, u_2 它指定相同的可能结果，即部分指定状态的集合 t，其中 $u \leq t$，对于 $u = u_1$, u_2而言是相同的。能够以不同方式处理这类部分指定状态，是基于操作的系统的一项优势 [2, 11, 21, 38]。这里我们将展示，在我们的系统中也可以实现这种控制。这可以通过修改 ℓ_{task} 中的过滤透镜来实现。

定义域 一种自然的做法是让筛选$_{\neg\mathrm{Done}}$/筛选$_{\mathrm{Apr1}}$ 将完成/推迟请求转换为添加请求。为此，我们对过滤器结果使用不同的定义域：$\mathbf{DT}_{\mathrm{OG}}$ 和 $\mathbf{DT}_{\mathrm{DT}}$，它们分别支持完成请求和推迟请求。我们重点讨论 $\mathbf{DT}_{\mathrm{OG}}$ ，因为 $\mathbf{DT}_{\mathrm{DT}}$ 的定义与此类似。我们将 Δ_{OG} 定义为由三元组（A， C， D）组成的集合，其中 A， $C \in$ Tasks 是任务的集合，而 $D \subseteq$编号是标识符的集合，并满足以下条件：

– $A(k) =$ (假, ,) 对所有 $k \in$dom(A)， – $C(k) =$ (是, ,) 对所有 $k \in$dom(C)，并且– dom(A)、dom(C)和 D 两两不相交。

直观地说，前两个条件表示， A 是视图中可见任务的添加请求，而 C 是不可见任务的添加请求。第三个条件断言了可行性。然后，我们使用以下 $\leq$ 和I定义 $\mathbf{DT}_{\mathrm{OG}} \triangleq \Delta_{\mathrm{OG}}$。

$$\frac{A \subseteq A' \quad C \subseteq C' \quad D \subseteq D'}{(A,C,D) \leq (A',C',D')} \qquad \frac{A \subseteq t \quad \mathrm{Disj}\,(\mathrm{dom}(C) \cup D, \mathrm{dom}(t))}{(A,C,D) \leq t}$$

$$\frac{A \subseteq A' \quad C \subseteq C' \quad D \subseteq D'}{(A,C,D) \in I_{(A',C',D')}} \qquad \frac{A \subseteq t}{(A,\emptyset,\emptyset) \in I_t}$$

在第 2.4 节中， A、 C和 D 分别由表中的（+）、（✓）和（–）标记表示。这个 $\mathbf{DT}_{\mathrm{OG}}$ 可与按照第 4.1 节类似定义的 $\oplus$ 复制，但由于我们不使用 dup 作为域，因此省略相关讨论。

修改后的 筛选 ¬已完成 对筛选$_{\neg\mathrm{Done}}$的修改很直接。现在，其视图域为 $\mathbf{DT}_{\mathrm{OG}}$（因此筛选$_{\neg\mathrm{Done}} \in$透镜$^{\leq}$ DT $\mathbf{DT}_{\mathrm{OG}}$）。其在严格部分指定状态上的行为发生了变化：$get$ 和put 现在利用 $A \in$ 任务$_{\mathrm{OG}}$ 和 $C \in$任务$_{\mathrm{Comp}}$这一事实，将 DT 中的添加请求 $A \uplus C$ 相互转换为由 $\mathbf{DT}_{\mathrm{OG}}$中的添加请求 A 和完成请求 C 组成的二元组。

$$get\ filter_{\neg\mathrm{Done}}\ (A \uplus C, D) = (A, C, D) \text{ where } A \in \mathsf{Tasks}_{\mathrm{OG}}, C \in \mathsf{Tasks}_{\mathrm{Comp}}$$
$$put\ filter_{\neg\mathrm{Done}}\ (_, (A, C, D)) = (A \uplus C, D)$$

（筛选$_{\mathrm{Apr1}} \in$ 透镜$^{\leq}$ DT $\mathbf{DT}_{\mathrm{DT}}$ 的定义与此类似）。此版本的良好行为性也可用与前一版本类似的方法证明（附录 A.5）。

5 将部分状态透镜视为基于操作的系统

本节展示了我们可以将更新编码为 i-偏序集，从而提供基于操作的系统的一项关键优势：对反向行为进行细粒度控制。本文中出现的 i-偏序集基于该方案构造，但存在一些细微偏差。

5.1 状态—更新对

构造 i-偏序集的基本思想是将更新与其来源配对，这一思想受到将基于操作的 CRDT 编码为基于状态的 CRDT 的启发 [44]。设 S 为一组（合法）状态，且 U 为更新的偏序集。[10] 我们假设 U 具有一个合并算子 $\oplus_U$，能够可靠地实现 U中的并，即，$u_1 \oplus_U u_2 = u$ 蕴含 $u_1 \vee u_2 = u$。我们还假设每个 $u \in U$ 都可以解释为一个偏函数 $[\![u]\!] \in S \rightharpoonup S$，直观上表示该更新的应用。令 $\mathsf{ran}_s(u) = \{s' \in S \mid \exists u'.\, u \leq_U u', [\![u']\!](s) = s'\}$，其直观含义是：当原始状态为 s时，u 可能产生的结果集合。然后，我们定义一个 i-偏序集 $\mathsf{G}_{S,U} = (S \cup (S \times U), \leq, I)$，其中 $\leq$和I定义如下：

$$\frac{u \leq_U u'}{(s,u) \leq (s,u')} \qquad \frac{}{s \leq s} \qquad \frac{s' \in \mathrm{ran}_s(u)}{(s,u) \leq s'} \qquad \frac{u \leq_U u'}{(s,u) \in I_{(s,u')}} \qquad \frac{}{s \in I_s} \qquad \frac{[\![u]\!](s) = s}{(s,u) \in I_s}$$

注意，$\leq$ 是一个偏序，I 是($\leq$)的一个自反子集。

为了支持 ps-发起器，我们接着定义$(\cdot) \in \mathsf{G}_{S,\ U} \to S \rightharpoonup S$ 如下。

$$s \cdot _ = s \qquad (s,u) \cdot s' = [\![u]\!](s) \text{ if } s = s'$$

根据$(\cdot)$、($\leq$)和I的定义，可以推出这个$(\cdot)$的 u-一致性和 u-可接受性。

要在 $\mathsf{G}_{S,\ U}$ 上支持dup，还需要一些额外结构。我们首先定义合并算子 $\oplus_{S,\ U} \in \mathsf{G}_{S,\ U} \times \mathsf{G}_{S,\ U} \rightharpoonup \mathsf{G}_{S,\ U}$ 如下。

$$s \oplus_{S,U} s' = s \text{ if } s = s' \qquad s' \oplus_{S,U} (s,u) = (s,u) \oplus_{S,U} s' = s' \text{ if } s' \in \mathrm{ran}_s(u)$$
$$(s,u) \oplus_{S,U} (s',u') = (s, u \oplus_U u') \text{ if } s = s'$$

如果 $\oplus_U$ 满足 $\mathsf{ran}_s(_)$，则只要$u_1 \oplus_U u_2 = u$，$\mathsf{ran}_s(u_1) \cap \mathsf{ran}_s(u_2) \subseteq \mathsf{ran}_s(u)$。（根据 $\mathsf{ran}_s(_)$ 的定义，反向包含关系显然成立。）

引理 9。 *$\mathsf{G}_{S,\ U}$ 在合并算子 $\oplus_{S,\ U}$ 下是可复制的，当以下条件成立时：*

（G1）$\oplus_U$ 对任意 s，满足 $\mathrm{ran}_s(_)$，（G2）$\oplus_U$ 对任意 u，在 $\{u' \mid u' \leq_U u\}$ 上是全定义的，并且（G3）$\oplus_U$ 对任意 s，在 $\{u \mid [\![u]\!](s) = s\}$ 上是全定义且封闭的。

证明（略）。我们使用 (G1) 来证明 $\oplus_{S,U}$ 能够正确实现并运算，并使用 (G2) 和 (G3) 来证明合并算子是全定义的，且在 I_x 下封闭（(G2) 适用于 $x = (s,u)$ 情形，(G3) 适用于 $x = s$ 情形）。具体而言，(G1) 用于确保 $(s, u \oplus_U u')$ 是 $\mathsf{G}_{S,U}$ 中 (s,u) 和 (s,u') 的上界中的最小元。□

[10] 不同于 Ahman 和 Uustalu，[2], Diskin等人 [11], 我们不使用索引集族来建模更新，因为我们关注的是更新的合并及其应用。使用索引集会使后者成为全定义的，但合并本质上是一个偏操作。

这些条件 (G1)、(G2) 和 (G3) 几乎是对 G$_{S,U}$ 可复制所需条件的直接翻译。事实上，(G1)、(G2) 和 (G3) 各自都是必要条件。

对于 (G1)，我们有以下不涉及 $\oplus_U$ 的充分条件：更新空间 U 必须足够精细，即任何具有共同可能结果的两个更新，都可以细化为一个同样共享该可能结果的更新。

引理 10。(G1) 成立，当对于所有 $s' \in$ 范围$_s(u_1) \cap$ 范围$_s(u_2)$，存在 u 使得 $u_1 \leq_U u$、$u_2 \leq_U u$ 和 $s' \in$范围$_s(u)$。□

(G2) 可由 $\oplus_U$ 的以下代数性质推出，而这一性质是理想的。

引理 11。(G2) 成立，当且仅当以下两个条件均成立：*(1)* 对任意可比较的 u_1 和 u_2，$\oplus_U$ 均有定义；*(2)* $\oplus_U$ 具有结合性，即 $u_1 \oplus (u_2 \oplus u_3) = u \Leftrightarrow (u_1 \oplus u_2) \oplus u_3 = u$。□

5.2 从部分指定状态中消除状态

将状态 s 与更新 u 配对效率不高，因为更新通常比状态小。在 $u \leq s'$ 的前提中使用 ran$(u) = \bigcup_s$ran$_s(u)$，通常会使 $\oplus_{S,U}$不健全，因为这违反了 (G1)，而 (G1) 也是 G$_{S,U}$ 可复制的必要条件。不过，这也意味着，如果我们不会对 G$_{S,U}$使用 dup，或者 $\oplus_{S,U}$ 满足 ran$(u) = \bigcup_s$ ran$_s(u)$，那么可以省略 s。第 4.1 节和第 4.5 节中的 i-偏序集 DT 和 DT$_{\mathrm{OG}}$ 便是后一类的例子。它们略微偏离了该方案；其 Is 比该方案更严格，并且 $(A, D) \cdot t$ 是全定义的，而该方案假设 $[\![(A, D)]\!]$ 是部分的。

当 $s \in$ ran(u) 蕴含 $[\![u]\!](s) = s$ for any u 时，此外，我们有 $(\leq) = I$。这解释了为什么在简单的 i-偏序集中，例如离散 i-偏序集（其中 U 为空，因此条件自然成立）、通过添加不同的最小元 Ω 从离散 i-偏序集构造而来的 i-偏序集（其中 ran$(\Omega) = S$ and$[\![\Omega]\!](s) = s$ for any s），以及将完整状态 s 与部分指定状态 $\{s\}$ 视为相同的 S 的幂集 i-偏序集（其中 ran$(T) = T$ 且 $[\![T]\!](s) = s$ if$s \in T$，否则对于某个元素 $t_s \in T$，$[\![T]\!](s) = t_s$），I 与 $(\leq)$ 相同。

6 相关工作

部分指定状态以各种形式出现在双向编程语言文献中。最初的透镜形式化 [13, 14] 使用 Ω 来表示不可用的源。然而，这种对 Ω的使用可以通过其他方式替代，例如$create$ 函数（例如 [6, 20]），该函数仅根据更新后的视图计算更新后的源。双向编程语言 HOBiT [36] 采用部分指定状态来处理值环境。粗略地说，它们将上下文中的项 $\Theta \vdash e :\ \sigma$ 解释为类型为 Θ 的值环境与类型为 σ的值之间的透镜，其中其put 执行仅指定

发生在 e中的变量的值。例如，对于 $\Theta = \{x : \mathsf{Int},\ y : \mathsf{Bool}\}$，$\Theta \vdash x : \mathsf{Int}$ 的 put 接收 1 并返回 $\{x \mapsto 1\}$，而 $\Theta \vdash y : \mathsf{Bool}$ 的put 接收 True 并返回 $\{y \mapsto \mathsf{True}\}$，仅指定所使用变量的值。这类值环境可以视为部分指定状态；这对于以可组合的方式解释复合表达式（如 (x, y)）至关重要，其中复合表达式可以看作多个视图的微观形式。它们的良好行为性是可组合的、基于偏序的，并且类似于我们的弱良定义性，不同之处在于它们具有极大性要求，该要求确保其get只能涉及完整状态。它们没有与 ps-稳定性相对应的规律，因为由于外部更新反映，它们的最终源本质上是离散的。这一差异反映了允许部分指定状态存在的位置和方式：在它们的框架中，部分指定状态只允许作为值环境，并用于解释上下文中的项。我们还注意到，它们的目标是保证经典良定性 [13, 14]；它们不支持通过源共享实现视图间更新传播。

Mu 等人也提出了类似的想法： [39], 他们使用特殊标签来标识已更新的部分。一种标签形式是 $*v$，它表示该部分已更新，并使其在他们的复制透镜的 put 执行中具有优先级。从这个意义上说，他们带有 $*$ 的值比不带 $*$ 的值指定得更充分。他们还针对结构化列表更新考虑了标签： $x^{+}{:}xs$ 和 $x^{-}{:}xs$ 分别表示插入和删除 x。[11] 通用方案（第 5.1 节）只需稍作修改，就能处理他们的带标签列表。由于可以从带标签列表中获取更新的来源，因此可将带标签列表视为状态与更新对。序则由其列表复制的反向行为（例如，合并）给出，即 $x \leq y \triangleq$ (合并 $x\ y = y$)。请注意，我们需要区分作为完整状态的列表与作为部分指定状态的列表。后者指定的列表多于其自身；例如，我们有 $[2,3] \leq 1^{+}{:}\ [2,3]$。由于他们的 合并 无法合并相同更新，例如 $1^{+}{:}\ 1^{-}{:}\ [\,]$ 和 $1^{-}{:}\ 1^{+}{:}\ [\,],$，因而不满足（G3），我们必须进一步限制 I，使其在与完整状态比较时只接受显然相同的更新，例如 $1 :\ [\,],$。标签由 ps-发起器的 put 移除。

称为 WPutGet [19]的性质也被用作一致性的替代性质，用于刻画存在多个视图时双向变换的行为。

$$put\ \ell\ (s_0, v) = s \implies put\ \ell\ (s_0, get\ \ell\ s) = s \qquad (\textsc{WPutGet})$$

WPutGet 的理念是，在存在多个视图时，视图更新 v 可能是“不完整的”，也就是说，它可能无法一致地更新所有副本；但我们可以调整更新后的视图，以便 $get\ \ell\ s$，从而一致地更新所有副本，并使它们的反向传播结果相同。该性质既不具备可组合性（见 [33,附录 B], ），也不表明用户更新得到了保留。我们并不保证该性质；事实上，

[11] 最初，符号 $\oplus$ 和 $\ominus$ 被用于 [39]。我们使用不同的符号，以避免与我们对 $\oplus$的使用产生混淆。

我们有一个不满足 WPutGet 的良态部分状态透镜（见附录 C）。

如第 3.1 节所述，我们的透镜不要求满足 PutPut 定律 [14] 。

$$put\ \ell\ (s_0, v_1) = s_1 \wedge put\ \ell\ (s_1, v_2) = s_2 \Longrightarrow put\ \ell\ (s_0, v_2) = s_2 \qquad (\textsc{PutPut})$$

除了满足一致性和可接受性之外，还满足该定律的经典透镜称为良好行为性透镜 [14]。尽管一些现有框架 [4,17,18,30,47] 要求在构造双向变换时满足 PutPut 定律，并在视图更新的语境下将其作为透镜范畴生成的基础 [8,43], ，但人们普遍认为，该定律过于严格，无法支持有用的变换 [14,26], ，例如数据库关系的投影。在我们的框架中，即使是部分状态初始化器等简单透镜，尤其是在视图允许“空操作”更新时，也不满足该定律。例如，对于一个部分状态初始化器 初始化 $\in$ 透镜$^{\leq}$ $\mathbb{N}\ \mathbb{N}_{\Omega}$，我们有：***put*** 初始化 (42，1) $= 1$ 和 ***put*** 初始化 (1，Ω) $= 1$，而 ***put*** 初始化 (42，Ω) $= 42$。

在数据库领域，Hegner [18] 讨论了涉及序的双向性定律。与我们的不同，他们所说的序直观上表示插入和删除。即使对于与序无关的子部分，这些定律也比经典透镜的良好行为性要求强得多。除了可接受性和一致性之外，他还施加了一些条件，使put 具有透明性 [17]: 直观而言，任何更新都可以撤销；同步时机（调用 put）无关紧要（PutPut）；而put (*s*, *v*)是否有定义，只取决于get s 和 v。通过进一步要求满足关于序的条件，Hegner [18] 证明了对于给定的get，put 是唯一的。

在部分指定状态中编码更新的通用方案（第 5.1 节）为我们提供了直接比较其他基于操作的框架的基础。Delta 透镜会同时正向和反向转换更新 [11]。遵循 Ahman 和 Uustalu [2], 的表述，Delta 透镜中的域是一个二元组 (*S*, *P*)，由状态集合 S 以及一族 $\{P_s\}_{s\in S}$ 更新集合组成，这些更新均始于 s。因此，(*S*, *P*) 与 (*V*, *Q*) 之间的 Delta 透镜 ℓ 是由三个函数构成的三元组：get $\ell \in S \to V$、dget $\ell \in (s : S) \to P\ s \to Q$(get s) 和 dput $\ell \in (s : S) \to Q$ (get s) $\to P\ s$。这里我们使用了 Agda 中表示依赖积的记法。即使以 Delta 透镜为基础展开讨论，我们也能得到类似的定律。采用这种方案的一个优点是，可以清晰地区分完整状态和严格部分指定状态。例如，只关注更新的某种良好行为性可以简单地表示为一个公式：dput $\ell\ s\ u \leq u' \Leftrightarrow u \leq$dget $\ell\ s\ u'$。然而，如果考虑 dup，这种区分也会使框架难以与经典透镜集成。给定 (*S*, $\lambda\ .S$) 表示一个基于状态的域 [2], ，与 ps-发起器对应的 Delta 透镜 init 会将 (*S*, $\lambda\ .S$) 转换为 (*S*, *P*)，其中 P取某个值。自然地，它的 dget 会将状态 s'嵌入为常量更新 $!s' \in P\ s$，即 dget init $s\ s' = !s'$；而它的 – dput 会取更新结果，即 dput init $s\ u = [\![u]\!](s)$。然而，统一定律随后会强制要求$[\![u]\!]$ (*s*) $= s' \Leftrightarrow u \leq !s'$。这要求过强，因为它要求 $P\ s$ 中任何可比较的更新都必须具有相同的目标状态。对于 $u_1 \leq u_2$，我们有 $u_2 \leq !\,[\![u_2]\!](s)$，因此 $u_1 \leq !\,[\![u_2]\!](s)$，进而推出 $[\![u_1]\!](s) = [\![u_2]\!](s)$。这实际上限制了 $P\ s$ 中的合并算子，使其只能合并目标状态相同的更新。我们的框架从设计上避免了这一问题：完整状态不携带来源信息，并且在通用方案中，$\leq$ 和 I 采用不同的处理方式。

更新-更新透镜 [2] 是 Delta 透镜的一种变体，只能反向转换更新。因此，它排除了仅由严格部分指定状态构成的域，例如无冲突复制数据类型（CRDT）[3, 44]。此外，在该框架中，我们只能通过比较严格部分指定状态与完整状态来陈述更新保持性。然而，这会大大削弱透镜在保持用户意图方面所能提供的保证；请注意，通常而言，更新所携带的信息比状态更多。例如，考虑第4.5节中的 $\mathrm{DT}_{\mathrm{OG}}$，其中，$(A, C, D)$ 的 $\mathsf{dom}(C)$ 和 D与完整状态无法区分，即 $\{t \mid (A, C, D) \leq t\} = \{t \mid (A, C', D') \leq t\}$，当 $\mathsf{dom}(C) \cup D = \mathsf{dom}(C') \cup D'$ 时成立。更新-更新透镜框架很难排除那些会扰乱此类细粒度意图的透镜。

在编辑镜头中，[21], 透镜会保留一个内部状态（$\in C$）；其 $\mathsf{eget} \in (U_S, C) \to (U_V, C)$ 和 $\mathsf{eput} \in (U_V, C) \to (U_S, C)$ 会在源与视图之间转换更新（U_S 在源上，U_V 在视图上），同时改变其内部状态。[12]对于 C，一个直接的选择是源类型 S；此时，编辑镜头可以用 delta 透镜的（简单类型版本）表示为 $\mathsf{eget}\,(u, s) = (\mathsf{dget}\ s\ u, [\![u]\!](s))$ 和 $\mathsf{eput}\,(u, s) = \mathsf{let}\ u' = \mathsf{dput}\ s\ u\ \mathsf{in}\ (u', [\![u']\!](s))$。不过，我们通常并不需要全部源信息来转换更新；也可以在 C 中存储 put 过程中丢失的信息。由于内部状态 C 具有这种灵活性，在我们的设定中讨论定律会比讨论 delta 透镜时更复杂。

这些基于操作的框架会考虑更新的复合，并要求其更新转换保持复合。这是 PutPut 定律的一种形式 [14], ，但与基于状态的情形不同，这项要求并不会使它们的透镜变得不切实际。在极端情况下，若更新以一系列原子更新的形式给出，那么只需逐一转换这些原子更新，就能满足组合保持要求。然而，在我们的设定中，它们所说的复合保持意义不大，因为更新后的视图可能不同于更新后的源所对应的视图，因此，对于一系列 $u_1 \fatsemi u_2$ 更新，传播 u_1 之后的视图状态可能不同于 u_2 所预期的视图。我们或许可以将操作表示为复合：将 $u_1 \leq u_2$ 视为 $u_2 = u_1 \fatsemi u$，其中 u，并通过推出定义更新的合并算子。然而，尽管这种观点能在我们的设定中给出一种 PutPut 定律，但对于一种基本的更新定义而言，这种方法并不奏效。在该定义中，更新由基本更新序列表示，且不同序列彼此有别。虽然可以考虑在这类序列上定义等式理论，以获得有意义的推出，但更合理的做法是假设更新具有范式，并在范式上执行操作；本文采用的正是这种方法。

[12] 原始编辑透镜是对称透镜的基于操作的版本 [20], ，但这里我们关注的是非对称情形。此外，$eget$ 和 $eput$ 并非原始名称；它们最初分别称为 $\Rightarrow$ 和 $\Leftarrow$ [21]。

7 结论

我们引入了部分状态透镜，将经典透镜 [13, 14] 扩展到部分指定状态。部分指定状态具有偏序关系，从而为更新保持和合并提供了清晰的概念。这对于处理共享同一源的多个视图至关重要。该框架的一项关键优势是支持对顺序感知良性进行组合推理，其能力足以保证用户更新得到保留。

未来的一个方向是为各种数据类型（如列表和有限映射）探讨更多原语和组合子。对于列表，我们将以 Mu 等人 [39]的列表为基础，正如第6节所述。另一个方向是提供一个前端系统，让用户能够将自己的意图表达为部分指定状态。一种方法是将所提出的框架与类 Elm 架构相连接，提供一个图形用户界面，以部分指定状态的形式发出更新。

致谢。 我们感谢 ZihangYe，他启发了本研究的方向。本研究部分得到了 JSPS KAKENHI Grant Numbers JP20H04161、JP23K20379、JP22H03562 和 JP23K24818，以及 EPSRC Grant*EXHIBIT*：用于双向变换的表达性高级语言 (EP/T008911/1)的资助。

利益披露。 作者声明不存在与本文内容相关的竞争性利益。

参考文献

1. Abou-Saleh, F., Cheney, J., Gibbons, J., McKinna, J., Stevens, P.：关于单子透镜的思考。载于：Lindley, S., McBride, C., Trinder, P.W., Sannella, D.（编）《改变世界的成功经验集——献给 Philip Wadler 60 岁生日的论文》。《计算机科学讲义》，第9600卷，第1–31页。施普林格（2016）。 https://doi.org/10.1007/978-3-319-30936-1_1, https://doi.org/ 10 1007 978 3 319 30936 1_1 。 /----。 Ahman, D., Uustalu, T.：[2]认真看待更新。载于：Eramo, R., Johnson, M.（编）第六届双向变换国际研讨会论文集，该研讨会与欧洲软件理论与实践联合会议 BX@ETAPS 2017 联合举办，瑞典乌普萨拉，2017年4月29日。CEUR研讨会论文集，第1827卷，第59–73页。CE UR-WS.org（2017），https://ceur-ws.org/Vol-1827/paper11.pdf 3. Almeida, P.S., Shoker, A., Baquero, C.：增量状态复制数据类型。《并行与分布式计算杂志》**111**，162–173（2018）。https://doi.org/10.1016/J.JPDC.2017. 08003 4003, 101016201708.https://doi.org/./j.jpdc...。 Bancilhon, F., Spyratos, N.：关系视图的更新语义。《ACM数据库系统汇刊》**6**(4)，557–575（1981）。 https://doi.org/10.1145/319628.319634，https: //doi.org/10319634 51145319628./.。 Barbosa, D.M.J., Cretin, J., Foster, N., Greenberg, M., Pierce, B.C.：匹配透镜：对齐与视图更新。载于：Hudak, P., Weirich, S.（编）第15届 ACM SIGPLAN 函数式编程国际会议论文集，ICFP 2010，美国马里兰州巴尔的摩，2010年9月27日至29日。第193–204页。ACM（2010）。https: //doi.org/10.1145/1863543.1863572, https://doi.org/10.1145/1863543.1863572

6. Bohannon, A., Foster, J.N., Pierce, B.C., Pilkiewicz, A., Schmitt, A.：Boomerang：用于字符串数据的资源型透镜。载于：Necula, G.C., Wadler, P.（编）第35届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL 2008，美国加利福尼亚州旧金山，2008年1月7日至12日。第407–419页。ACM（2008）。https://doi.org/10.1145/1328438.1328487, https://doi.org/ 10 1328487 7 1145 1328438 . / . . Bohannon, A., Pierce, B.C., Vaughan, J.A.：关系透镜：一种可更新视图语言。载于：Vansummeren, S.（编）第二十五届 ACM SIGACT-SIGMOD-SIGART 数据库系统原理研讨会论文集，2006年6月26日至28日，美国伊利诺伊州芝加哥。第338–347页。ACM（2006）。https://doi.org/10. 1145 1142399 8 1142351 1142399, 10 1145 1142351 5 / . https://doi.org/ . / . . Boisseau, G.：用于光学的字符串图。载于：Ariola, Z.M.（编）计算与演绎形式结构国际会议，FSCD 2020，2020年6月29日至7月6日，法国巴黎（虚拟会议）。LIPIcs，第167卷，第17:1–17:18页。Schloss Dagstuhl - Leibniz 信息学中心（2020）。https://doi. org/10 17 9 4230 2020 17, 10 4230 2020 . /LIPICS.FSCD. . https://doi.org/ . / LIPIcs.FSCD. . . Chin, W.：迈向自动元组化策略。载于：Schmidt, D.A.（编）ACM SIGPLAN 部分求值与基于语义的程序操作研讨会论文集，PEPM'93，1993年6月14日至16日，丹麦哥本哈根。第119–132页。ACM（1993）。https://doi.org/ . / . https: //doi.org/10 154643 10 1145 154630. / . . Cunha, J., Fernandes, J.P., Mendes, J., Pacheco, H., Saraiva, J.：模型驱动电子表格的双向变换。载于：Hu, Z., de Lara, J.（编）模型变换理论与实践——第五届国际会议，ICMT@TOOLS 2012，捷克共和国布拉格，2012年5月28日至29日。论文集。《计算机科学讲义》，第7307卷，第105–120页。施普林格（2012）。https://doi.org/10.

11 1007 978 3 642 30476 7_7 10 1007 978 3 642 30476 7_7 / - - - - , https://doi.org/ . / - - - - . Diskin, Z., Xiong, Y., Czarnecki, K.：从基于状态到基于增量的双向模型变换：非对称情形。《面向对象技术杂志》10，6: 1–25（2011）。https: //doi.org/10 6 12 5381 2011 10 1 6, 10 5381 2011 10 1 . /JOT. . . .A https://doi.org/ . /jot. . . .a . Fischer, S., Hu, Z., Pacheco, H.：透镜定律的清晰阐释——函数式编程珍珠。载于：Hinze, R., Voigtländer, J.（编）程序构造数学——第十二届国际会议，MPC 2015，2015年6月29日至7月1日，德国柯尼希斯温特。论文集。《计算机科学讲义》，第卷，第215–223页。施普林格（2015）。https://doi.org/10.1007/978-3-319-19797-5_10, https://doi.org/

13 10 1007 978 3 319 19797 5_10 . / - - - - . Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.：双向树变换的组合子：一种解决视图更新问题的语言学方法。载于：Palsberg, J., Abadi, M.（编）第32届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL 2005，美国加利福尼亚州长滩，2005年1月12日至14日。第233–246页。ACM（2005）。https: //doi.org/10 1040325 14 1145 1040305 1040325, 10 1145 1040305 . / . https://doi.org/ . / . . Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.：双向树变换的组合子：一种解决视图更新问题的语言学方法。《ACM程序设计语言与系统汇刊》29(3)（2007）15. Foster, J.N., Pilkiewicz, A., Pierce, B.C.：商透镜。载于：Hook, J., Thiemann, P.（编）第13届 ACM SIGPLAN 函数式编程国际会议论文集，ICFP 2008，2008年9月22日至28日，加拿大不列颠哥伦比亚省维多利亚。第383–396页。ACM（2008）。https://doi.org/ . / . https: //doi.org/10.1145/1411204.1411257

16. Goldstein, H., Frohlich, S., Wang, M., Pierce, B.C.：关于随机生成的思考。《ACM程序设计语言会议录》7(ICFP)，322–355（2023）。https://doi.org/10.1145/3607842，https://doi.org/10.1145/3607842 17. Hegner, S.J.：封闭数据库视图规范更新支持的基础。载于：Abiteboul, S., Kanellakis, P.C.（编）ICDT'90，第三届国际数据库理论会议，巴黎，法国，1990年12月12日至14日，论文集。《计算机科学讲义》，第470卷，第422–436页。施普林格（1990）。https://doi.org/ 10.1007/3-540-53507-1_93，https://doi.org/10.1007/3-540-53507-1_9318. Hegner, S.J.：封闭数据库视图更新的序理论。《人工智能与数学年刊》40(1-2)，63–125（2004）。https://doi.org/10.1023/A:1026158013113，https://doi.org/10.1023/A:1026158013113 19. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K., Nakano, K.：图变换的双向化。载于：Hudak, P., Weirich, S.（编）第15届 ACM SIGPLAN 函数式编程国际会议论文集，ICFP 2010，美国马里兰州巴尔的摩，2010年9月27日至29日。第205–216页。ACM（2010）。https: //doi.org/10.1145/1863543.1863573，https://doi.org/10.1145/1863543.1863573 20. Hofmann, M., Pierce, B.C., Wagner, D.：对称透镜。载于：Ball, T., Sagiv, M.（编）第38届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL 2011，美国得克萨斯州奥斯汀，2011年1月26日至28日。第371–384页。ACM（2011）。https://doi.org/10.1145/1926385.1926428，https://doi.org/10.1145/1926385.1926428 21. Hofmann, M., Pierce, B.C., Wagner, D.：编辑镜头。载于：Field, J., Hicks, M.（编）第39届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL 2012，美国宾夕法尼亚州费城，2012年1月22日至28日。第495–508页。ACM（2012）。https://doi.org/10.1145/2103656.2103715，https://doi.org/10.1145/2103656.2103715 22. Horn, R., Perera, R., Cheney, J.：增量关系镜头。《ACM程序设计语言会议录》2(ICFP)，74:1–74:30（2018）。https://doi.org/10.1145/3236769，https://doi. org/10.1145/3236769 23. Hu, Z., Iwasaki, H., Takeichi, M., Takano, A.：元组化计算消除多次数据遍历。载于：Peyton Jones, S.L., Tofte, M., Berman, A.M.（编）1997年 ACM SIGPLAN 函数式编程国际会议论文集（ICFP '97），荷兰阿姆斯特丹，1997年6月9日至11日。第164–175页。ACM（1997）。https://doi.org/10.1145/258948.258964，https://doi.org/10.1145/258948.258964 24. Hu, Z., Ko, H.：BiGUL 双向编程的原理与实践。载于：Gibbons, J., Stevens, P.（编）《双向变换》国际暑期学校，英国牛津，2016年7月25日至29日，教程讲座。《计算机科学讲义》，第9715卷，第100–150页。施普林格（2016）。https://doi.org/10. 1007/978-3-319-79108-1_4，https://doi.org/10.1007/978-3-319-79108-1_425. Hu, Z., Mu, S., Takeichi, M.：一种基于双向变换、用于开发结构化文档的可编程编辑器。载于：Heintze, N., Sestoft, P.（编）2004年 ACM SIGPLAN 部分求值与基于语义的程序操作研讨会论文集，2004，25，2004 24 178 189 2004 10 1145 1014007 1014025，意大利维罗纳，8月 - 。第 – 页。ACM（）。https://doi.org/ . / . https: //doi.org/10.1145/1014007.1014025 26. Keller, A.M.：关于Bancilhon和Spyratos的"更新语义与关系视图"的评论。《ACM数据库系统汇刊》12(3)，521–523（1987）。https: //doi.org/10.1145/27629.214296，https://doi.org/10.1145/27629.214296

27. Ko, H., Zan, T., Hu, Z.：Bigul：一种形式化验证的核心语言，用于基于putback的双向编程。载于：Erwig, M., Rompf, T.（编）2016年 ACM SIGPLAN 部分求值与程序操作研讨会论文集，PEPM 2016，22，2016 20 61 72 2016 10 1145 2847538 2847544，圣彼得堡，美国佛罗里达州，1月 - 。第 – 页。ACM（）。https://doi.org/ . / . http: //doi.acm.org/10 2847544 28 1145 2847538 . / . . van Laarhoven, T.：基于CPS的函数式引用。博客文章：https://www.twanvl. nl/blog/haskell/cps-functional-references（2009 08 29 2024 10 ），访问日期 - - 。Macedo, N., Pacheco, H., Sousa, N.R., Cunha, A.：双向电子表格公式。载于：Fleming, S.D., Fish, A., Scaffidi, C.（编）IEEE 可视化语言与以人为中心的计算研讨会，VL/HCC 2014，澳大利亚维多利亚州墨尔本，2014年7月28日至8月1日。第161–168页。IEEE 计算机学会（2014）。https://doi.org/ 10 6883041 30 1109 2014 6883041，10 1109 2014 . /VLHCC. . https://doi.org/ . /VLHCC. . . Matsuda, K., Hu, Z., Nakano, K., Hamana, M., Takeichi, M.：基于视图补全函数自动推导的双向化变换。载于：Hinze, R., Ramsey, N.（编）第12届 ACM SIGPLAN 函数式编程国际会议论文集，ICFP 2007，德国弗赖堡，2007年10月1日至3日。第47–58页。ACM（2007）。https://doi.org/10.1145/1291151.1291162, https: //doi.org/10 1291162 31 1145 1291151 . / . . Matsuda, K., Wang, M.：FliPpr：一种更美观的可逆打印系统。载于：Felleisen, M., Gardner, P.（编）ESOP。《计算机科学讲义》，第7792卷，第101–120页。施普林格（2013）。https://doi.org/10.1007/978-3-642-37036-6_6，https: //doi.org/ 10 1007 978 3 642 37036 6_6 . / - - - - 。Matsuda, K., Wang, M.：使用透镜进行应用式双向编程。载于：Fisher, K., Reppy, J.H.（编）ICFP。第62–74页。ACM（2015）。https://doi.org/10. 1145 2784750 33 2784731 2784750，10 1145 2784731 / . http://doi.acm.org/ . / . . Matsuda, K., Wang, M.：“免费双向化”用于单态变换。《计算机科学与程序设计》111，79–109（2015）。https://doi.org/10.1016/j. scico.2014 008 34 07 008，10 1016 2014 07 . . https://doi.org/ . /j.scico. . . . Matsuda, K., Wang, M.：应用式双向编程：混合透镜与语义双向化。《函数式编程杂志》28，e15（2018）。https://doi.org/ 10 0956796818000096 35 1017 0956796818000096，10 1017 . /S https://doi.org/ . /S . Matsuda, K., Wang, M.：嵌入含绑定子的可逆语言：FliPpr 语言的一个案例。载于：Wu, N.（编）第11届 ACM SIGPLAN Haskell 国际研讨会论文集，Haskell@ICFP 2018，美国密苏里州圣路易斯，2018年9月27日至17日。第158–171页。ACM（2018）。https://doi.org/10.1145/3242744. 3242758，3242758 36 10 1145 3242744 https://doi.org/ . / . . Matsuda, K., Wang, M.：HOBiT：无需使用透镜组合子的编程透镜。载于：Ahmed, A.（编）ESOP。《计算机科学讲义》，第10801卷，第31–59页。施普林格（2018）。https://doi.org/10.1007/978-3-319-89884-1_2，https://doi.org/ 10 1007 978 3 319 89884 1_2 . / - - - - 。Mayer, M., Kuncak, V., Chugh, R.：通过直接操作进行双向求值。《ACM程序设计语言会议录》2(OOPSLA)，127:1–127:28（2018）。https: //doi.org/10 3276497 38 1145 3276497，10 1145 . / https://doi.org/ . / . Meertens, L.：为用户交互设计约束维护器（1998），可在以下网址获取：https://www.kestrel.edu/people/meertens/pub/dcm.pdf 39. Mu, S., Hu, Z., Takeichi, M.：双向更新的代数方法。载于：Chin, W.（编）《编程语言与系统：第二届亚洲研讨会》，APLAS 2004，中国台北，2004年11月4日至6日，论文集。《计算机科学讲义》，第3302卷，第2–20页。施普林格（2004）。https://doi.org/10.1007/ 978-3-540-30477-7_2, https://doi.org/10.1007/978-3-540-30477-7_2

40. O'Connor, R.：函子之于透镜，犹如应用函子之于双板：引入多板。CoRR abs/1103.2841（2011），http://arxiv.org/abs/1103.2841，已被 WGP '11 接收，但未收录于其论文集 41. Pacheco, H., Cunha, A.：泛型无点透镜。载于：Bolduc, C., Desharnais, J., Ktari, B.（编）程序构造数学，第10届国际会议 MPC 2010，魁北克市（加拿大），2010年6月21日至23日。论文集。《计算机科学讲义》，第6120卷，第331–352页。施普林格（2010）。https://doi.org/ 10 1007 978 3 642 13321 3_19 10 1007 978 3 642 13321 3_19 . / - - - - 42, https://doi.org/ . / - - - - . Pickering, M., Gibbons, J., Wu, N.：Profunctor 光学：模块化数据访问器。Art Sci. Eng. Program. 1(2), 7 (2017). https://doi.org/10.22152/PROGRAMMING-JOURNAL.ORG/2017/1/7, https://doi.org/10.22152/programming-journal.org/2017 7 43 1 2018/ / . Riley, M.：光学范畴 ( ), https://arxiv.org/abs/1809 00738 44 . . Shapiro, M., Preguiça, N.M., Baquero, C., Zawirski, M.：无冲突复制数据类型。载于：Défago, X., Petit, F., Villain, V.（编）分布式系统的稳定性、安全性与安全保障——第13届国际研讨会 SSS 2011，格勒诺布尔（法国），2011年10月10日至12日。论文集。《计算机科学讲义》，第6976卷，第386–400页。施普林格（2011）。https://doi.org/10.1007/978-3-642-24550-3_29, https: //doi.org/ 10 1007 978 3 642 24550 3_29 . / - - - - . Stevens, P.：双向模型转换概览。载于：Lämmel, R., Visser, J., Saraiva, J.（编）GTTSE。《计算机科学讲义》，第5235卷，第408–424页。施普林格（2008）。https://doi.org/10.1007/978-3-540-88643-3_10, https: //doi.org/ 10 1007 978 3 540 88643 3_10 . / - - - - . Takeichi, M.：用函数配置双向程序。发表于 IFL 2009：函数式语言实现与应用国际研讨会／工作坊（2009），可从以下网址获取：https://takeichimasato.net/ attachments/XFun0.pdf 47. Voigtländer, J.：免费实现双向化！（珍珠）。载于：Shao, Z., Pierce, B.C.（编）第36届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL 2009，2009年1月23日，萨凡纳（佐治亚州，美国）。第165–176页。ACM（2009）。https://doi.org/ . / . https: //doi.org/10 1480904 48 1145 1480881 . / . . Voigtländer, J., Hu, Z., Matsuda, K., Wang, M.：结合句法与语义双向化。载于：Hudak, P., Weirich, S.（编）第15届 ACM SIGPLAN 函数式编程国际会议论文集，ICFP 2010，美国马里兰州巴尔的摩，2010年9月27日至29日。第181–192页。ACM（2010）。https: //doi.org/10 1863571 49 1145 1863543 1863571, 10 1145 1863543 . / . https://doi.org/ . / . . Voigtländer, J., Hu, Z., Matsuda, K., Wang, M.：通过形状双向化器插件增强语义双向化。《函数式编程杂志》23(5)，515–551（2013）。https://doi.org/10.1017/S0956796813000130, https://doi.org/10.1017/S0956796813000130 50. Williams, J., Gordon, A.D.：电子表格双向编辑的 where 来源追踪。《计算机语言杂志》73，101155（2022）。https://doi.org/10.1016/J.COLA.2022. 101155, 101155 51 10 1016 2022 https://doi.org/ . /j.cola. . . Xie, R., Schrijvers, T., Hu, Z.：双向解析器：用于数据同步的精确打印。ACM程序设计语言会议录 9(POPL)，2205–2231（2025）。https://doi.org/10.1145/ 3704910, 3704910 52 10 1145 https://doi.org/ . / . Xiong, Y., Liu, D., Hu, Z., Zhao, H., Takeichi, M., Mei, H.：从模型转换实现自动模型同步。载于：Stirewalt, R.E.K., Egyed, A., Fischer, B.（编）第22届 IEEE/ACM 自动化国际会议

软件工程（ASE 2007），2007年11月5日至9日，美国佐治亚州亚特兰大。第164–173页。ACM（2007）。https://doi.org/10.1145/1321631.1321657，https://doi.org/101321657 5311451321631./.。Yu, Y., Lin, Y., Hu, Z., Hidaka, S., Kato, H., Montrieux, L.：通过双向变换维护不变量可追溯性。载于：Glinz, M., Murphy, G.C., Pezzè, M.（编）第34届国际软件工程会议，ICSE 2012，2012年6月2日至9日，瑞士苏黎世。第540–550页。IEEE 计算机学会（2012）。https://doi.org/10.1109/ICSE.2012.6227162，https://doi.org/10.1109/ICSE.2012.622716254. Zhang, X., Hu, Z.：面向不完整程序的双向实时编程。载于：第44届 IEEE/ACM 第44届国际软件工程会议，ICSE 2022，美国宾夕法尼亚州匹兹堡，2022年5月25日至27日。第2154–2164页。ACM（2022）。https: //doi.org/103510195 551145351000335101 95，1011453510003./.https://doi.org/./.。Zhang, X., Xie, R., Guo, G., He, X., Zan, T., Hu, Z.：将直接操纵融合到函数式程序中。ACM程序设计语言会议录 8(POPL)，1211–1238（2024）。https://doi.org/103632883 5611453632883，101145./https://doi.org/./。Zhu, Z., Ko, H., Martins, P., Saraiva, J., Hu, Z.：Biyacc：将解析器和反射式打印器合二为一。载于：Cunha, A., Kindler, E.（编）第四届国际双向变换研讨会论文集（与软件技术：应用与基础会议 STAF 2015 联合举办），2015年7月24日，意大利拉奎拉。CEUR研讨会论文集，第1396卷，第43–50页。CEUR-WS.org（2015），https://ceur-ws.org/Vol-1396/p43-zhu.pdf 57. Zhu, Z., Zhang, Y., Ko, H., Martins, P., Saraiva, J., Hu, Z.：双向解析与反射式打印。载于：va n der Storm, T., Balland, E., Varró, D.（编）2016年 ACM SIGPLAN 国际软件语言工程会议论文集，荷兰阿姆斯特丹，2016年10月31日至11月1日。第2–14页。ACM（2016）。https://doi.org/10.1145/2997364，http://dl.acm.org/citation.cfm?id=2997369

A 证明

A.1 引理 2

设 v 为$get\ \ell\ s$。假设 $s \leq s'$。根据 ps-可接受性，$put\ \ell\ (s,\ v)$ 有定义，并产生某个 s''，使得 $s'' \in I_s$。由于I是 $\leq$的子集，因此我们有 $s'' \leq s$。然后，根据 $s'' \leq s \leq s'$ 的 ps-一致性，我们有 $v \leq get\ \ell\ s'$。

A.2 引理 3

设 v 为$get\ \ell\ s$。

根据 ps-可接受性，$put\ \ell\ (s,\ v)$ 必须有定义；设其结果为 s'。这也确保了 $s' \in I_s$，由此可推出 $s' \leq s$。接着，我们对$put\ \ell\ (s,\ v) = s'$ 和 $s' \leq s'$应用 ps-一致性，得到 $v \leq get\ \ell\ s'$。然后，根据 get 的单调性（引理 2）以及 $s' \leq s$，可得 $get\ \ell\ s' \leq get\ \ell\ s = v$。由于我们同时有$v \leq get\ \ell\ s'$ 和 $get\ \ell\ s' \leq v$，根据反对称性可得 $get\ \ell\ s' = v$。

A.3 引理 4

我们先证明，对于任意 $v \in V_s$，都有 $v \leq get\ \ell\ s$，然后证明 $get\ \ell\ s \in V_s$。设 v 是 V_s 的一个元素，即存在某些 s_0 和 s_1，使得有 $put\ \ell\ (s_0,\ v) = s_1$ 和 $s_1 \leq s$。根据 ps-一致性，可得 $v \leq get\ \ell\ s$。接着，设 $get\ \ell\ s$ 为 v。根据 ps-可接受性，对于某个满足 $s' \in I_s$ 的 s'，有 $put\ \ell\ (s,\ v) = s'$，这意味着$s' \leq s$。因此，我们有 $get\ \ell\ s \in V_s$。

A.4 第 4.3 节中 筛选$_{\neg\mathrm{Done}}$ 的良好行为性

请注意，筛选$_{\neg\mathrm{Done}}$ 的定义保证了当只涉及完整状态时，它在经典意义下是良态的，因此我们只需考察涉及严格部分指定状态的情况。

首先，我们确认 ps-一致性。假设 ***put*** 过滤器$_{\neg\mathrm{Done}}(s, v) = s'$。由于 ***get*** 过滤器 $_{\neg\text{已完成}}$ 是单调的，只需证明 $v \leq$ ***get*** 过滤器$_{\neg\mathrm{Done}}\ s'$。通过对 v 分情况讨论，容易根据筛选 $_{\neg\text{已完成}}$的定义得出 $v =$ ***get*** 过滤器$_{\neg\mathrm{Done}}\ s'$。

其次，我们确认 ps-可接受性。假设 get 过滤器$_{\neg\mathrm{Done}}\ s = v$且 $v' \in \mathrm{I}_v$。我们将证明，对于某个满足 $s' \in \mathrm{I}_s$ 的 s'，有 put 过滤器$_{\neg\mathrm{Done}}(s, v') = s'$。考虑 $s = t \in$ 任务的情况。如果 $v' = v = t_{\mathrm{OG}}$，则容易看出 $s' = (t \setminus t_{\mathrm{OG}}) \lhd t_{\mathrm{OG}} = t$，从而确保 $s' = t \in \mathrm{I}_t$。如果 $v' = (A, D) \in \mathrm{I}_{t_{\mathrm{OG}}}$，那么必然是 $A \subseteq t_{\mathrm{OG}} \in$任务$_{\mathrm{OG}}$ 且 $D = \emptyset$，这意味着$s' = (A, \emptyset)$。由于已知 $A \subseteq t_{\mathrm{OG}} \subseteq t$，因此有 $s' = (A, \emptyset) \in \mathrm{I}_t$。考虑 $s = (A, D)$ 的情况。那么，必然有 $v = (A_{\mathrm{OG}}, D)$。根据定义，产生最大 s' 的 v' 是 $v' = v$，因此只需考虑 $v' = v$ 的情况。在这种情况下，$s' = (A_{\mathrm{OG}}, D)$。由于已知 $A_{\mathrm{OG}} \subseteq A$，因此有$s' = (A_{\mathrm{OG}}, D) \in \mathrm{I}_{(A,D)}$。

最后，我们确认 ps-稳定性。假设 ***put*** 过滤器$_{\neg\mathrm{Done}}(s_0,\ v) = s$，且 ***put*** 过滤器 $_{\neg\mathrm{Done}}(s',\ v'') = s''$满足 $s \leq s'$和 $v \leq v'' \in I^{get\ filter_{\neg\mathrm{Done}}\ s'}$。我们

将证明 $s \leq s''$。当 s 是完整状态时，讨论中只涉及完整状态。因此，我们考虑这样的情形： $s = v = (A, D)$，且有$A \in$任务$_{\mathrm{OG}}$。当 $s' = (A', D')$ 时，get 过滤器 $_{\neg\mathrm{Done}}$ s' 也是严格部分指定状态。对于严格部分指定状态 v_1, v_2 ，若满足 $v_1 \leq v_2$，则有 put 过滤器$_{\neg\mathrm{Done}}(s', v_2) = s_2$意味着 put 过滤器 $\neg$已完成 $(s', v_1) \leq s_2$。于是，只需讨论 $v'' = v$ 的情形，因为它给出最小的 s''。在这种情况下，我们有$s'' = (A, D) = s$，因此 $s \leq s''$。当 s'是完整状态时，get 过滤器$_{\neg\mathrm{Done}}$ s'也是完整状态。如果 v'' 是完整状态，即 v'' = get 过滤器$_{\neg\mathrm{Done}}$ s'。那么，我们有 $s'' = (s' \setminus s'_{\mathrm{OG}}) \lhd s'_{\mathrm{OG}} = s'$，这意味着 $s \leq s'$。如果 v'' 是部分指定状态，则同样只需考虑 $v'' = v$ 的情形，由此可得$s'' = (A, D) = s$ ，因此 $s \leq s''$。

A.5 第4.5节中筛选$_{\neg\mathrm{Done}}$ 的良好行为性

由于筛选$_{\neg\mathrm{Done}}$ 在只涉及完整状态时仍然是经典意义上的良态，因此我们只需考察涉及严格部分指定状态的情况。

首先，我们确认 ps-一致性。假设*put* 过滤器$_{\neg\mathrm{Done}}(s, v) = s'$。由于*get* 过滤器 $_{\neg\text{已完成}}$ 是单调的，只需证明 $v \leq$*get* 过滤器$_{\neg\mathrm{Done}}$ s'。通过对 v进行分情况讨论，根据筛选 $_{\neg\text{已完成}}$的定义，很容易得出 $v =$ *get* 过滤器$_{\neg\mathrm{Done}}$ s' 。

其次，我们确认 ps-可接受性。假设 get 过滤器$_{\neg\mathrm{Done}}$ $s = v$且 $v' \in \mathrm{I}_v$。我们将证明，对于某个 s'满足 $s' \in \mathrm{I}_s$，有 put 过滤器$_{\neg\mathrm{Done}}(s, v') = s'$。考虑 $s = t \in$ 任务的情况。如果 $v' = v = t_{\mathrm{OG}}$，那么很容易看出 $s' = (t \setminus t_{\mathrm{OG}}) \lhd t_{\mathrm{OG}} = t$，从而确保 $s' = t \in \mathrm{I}_t$。如果 $v' = (A, D) \in \mathrm{I}_{t_{\mathrm{OG}}}$，那么必定是 $A \subseteq t_{\mathrm{OG}} \in$任务$_{\mathrm{OG}}$ 且 $D = \emptyset$，这意味着$s' = (A, \emptyset)$。由于我们有 $A \subseteq t_{\mathrm{OG}} \subseteq t$，因此有 $s' = (A, \emptyset) \in \mathrm{I}_t$。考虑 $s = (A, D)$的情况。那么必定有 $v = (A_{\mathrm{OG}}, D)$。根据定义，使 v' 达到最大值的 s' 是 $v' = v$，因此只需考虑 $v' = v$的情况。在这种情况下， $s' = (A_{\mathrm{OG}}, D)$。由于我们有 $A_{\mathrm{OG}} \subseteq A$，因此有$s' = (A_{\mathrm{OG}}, D) \in \mathrm{I}_{(A,D)}$。

最后，我们验证 ps-稳定性。假设 put 过滤器$_{\neg\mathrm{Done}}(s_0, v) = s$，且 put 过滤器 $_{\neg\mathrm{Done}}(s', v'') = s''$，其中 $s \leq s'$和 $v \leq v'' \in \mathrm{I}_{get\ filter_{\neg\mathrm{Done}}\ s'}$。我们将证明 $s \leq s''$。当 s 是完整状态时，讨论中只涉及完整状态。因此，我们考虑 $s = v = (A, D)$ with$A \in$任务$_{\mathrm{OG}}$的情况。当 $s' = (A', D')$时，get 过滤器$_{\neg\mathrm{Done}}$ s' 也是严格部分指定状态。对于满足 $v_1 \leq v_2$的严格部分指定状态 v_1、 v_2 ，有 put 过滤器$_{\neg\mathrm{Done}}(s', v_2)$ $= s_2$ implies put 过滤器 $\neg$已完成 $(s', v_1) \leq s_2$。于是，只需讨论 $v'' = v$ 的情况，因为它给出了最小的 s''。在这种情况下，我们有$s'' = (A, D) = s$，因此 $s \leq s''$。当 s'是完整状态时，get 过滤器$_{\neg\mathrm{Done}}$ s'也是完整状态。如果 v'' 是完整状态，即 v'' = get 过滤器$_{\neg\mathrm{Done}}$ s'。那么，我们有 $s'' = (s' \setminus s'_{\mathrm{OG}}) \lhd s'_{\mathrm{OG}} = s'$，这意味着 $s \leq s'$。如果 v'' 是部分指定状态，则同样只需考虑 $v'' = v$的情况，由此得到$s'' = (A, D) = s$ ，因此 $s \leq s''$。

A.6 引理 10

假设以下条件成立：对于所有 $s' \in \mathsf{ran}_s(u_1) \cap \mathsf{ran}_s(u_2)$，存在 u 使得 $u_1 \leq_U u$、$u_2 \leq_U u$ 且 $s' \in \mathsf{ran}_s(u)$。再假设 $u_1 \oplus_U u_2 = u_{12}$ 且 $s' \in \mathsf{ran}_s(u_1) \cap \mathsf{ran}_s(u_2)$。那么，由于 $\oplus_U$ 能够正确实现并运算，我们有 $u_{12} = u_1 \vee u_2$，从而有 $u_{12} \leq_U u$，因为并是 u_1 和 u_2 的最小公共上界。根据定义，$\mathsf{ran}_s(u)$ 是反单调的：对于任意 u 和 u'，$u \leq_U u'$ 意味着 $\mathsf{ran}_s(u') \subseteq \mathsf{ran}_s(u)$。因此，根据 $s' \in \mathsf{ran}_s(u)$ 和 $u_{12} \leq_U u$，我们也有 $s' \in \mathsf{ran}_s(u_{12})$，这证明了对于任意 s，$\oplus_U$ 都满足 $\mathsf{ran}_s(-)$。

A.7 引理 11

假设对于任意 u_1 和满足 $u_1 \leq u_2$ 的 u_2，$u_1 \oplus_U u_2$ 均有定义；并且对于任意 u_1, u_2, u_3, $u \in U$，$u_1 \oplus_U (u_2 \oplus_U u_3) = u$ 蕴含 $(u_1 \oplus_U u_2) \oplus_U u_3 = u$，反之亦然。注意，$u_1 \oplus_U u_2 = u$ 对于 $u_1 \leq u_2$ 蕴含 $u = u_2$。设 u_1 和 u_2 是满足 $u_1 \leq_U u$ 且对于某个 u 有 $u_2 \leq_U u$ 的更新。接下来，我们证明 $u_1 \oplus_U u_2$ 必须有定义。我们利用以下事实：根据假设 $(\trianglelefteq)$，$u_1 \oplus_U u = u$ 且 $u_2 \oplus_U u = u$ $\mathsf{dom}(\oplus_U)$。我们有：

$$
\begin{aligned}
u_1 \oplus_U u = u \Leftrightarrow & \{\ u_2 \oplus_U u = u \text{ by assumption } \} \\
& u_1 \oplus_U (u_2 \oplus_U u) = u \\
\Leftrightarrow & \{\ \text{associativity } \} \\
& (u_1 \oplus_U u_2) \oplus_U u = u
\end{aligned}
$$

由于 $(u_1 \oplus_U u_2) \oplus_U u$ 已定义，$u_1 \oplus_U u_2$ 也必须已定义。

请注意，我们还有 $u_1 \oplus_U u_2 \leq u$。

B 一个合法透镜，其 *put* 在第一个参数上不单调

考虑以下透镜 $\ell_{\text{putNonmono1}} \in \mathsf{Lens}^{\trianglelefteq}$ 布尔值$_\Omega$ $\mathbf{1}_\Omega$。

$$
\mathit{get}\ \ell_{\text{putNonmono1}}\ s = \begin{cases} \Omega & \text{if } s = \Omega \\ () & \text{otherwise} \end{cases}
$$

$$
\mathit{put}\ \ell_{\text{putNonmono1}}\ (s, v) = \begin{cases} \Omega & \text{if } v = \Omega \\ s & \text{if } v \neq \Omega \wedge s \neq \Omega \\ \mathsf{True} & \text{if } v \neq \Omega \wedge s = \Omega \end{cases}
$$

这个透镜在第一个参数上不单调，因为 $\mathit{put}\ \ell_{\text{putNonmono1}}\ (\Omega, ())$ =是，而 $\mathit{put}\ \ell_{\text{putNonmono1}}$ (假, ()) = 假，其中 是 $\trianglelefteq$ 假。请注意，这个 *get* 是单调的，并且 *put* 在第二个参数上也单调。

让我们确认这个透镜是良态的。首先，确认 ps-可接受性。由于其 *put* 在第二个参数上单调，只需检查 $\mathit{put}\ \ell_{\text{putNonmono1}}(s, \mathit{get}\ \ell_{\text{putNonmono1}}\ s) \in I_s$。当 $s = \Omega$ 时，左侧根据定义也为 Ω，满足 $\Omega \in I_\Omega$。当 $s \neq \Omega$ 时，左侧根据定义为 s，

满足 $s \in \mathrm{I}_s$。接下来，确认 ps-一致性。由于其 get 是全定义且单调的，只需检查 $v \leq \textsf{get}\ \ell_{\mathrm{putNonmono1}}\ (\textsf{put}\ \ell_{\mathrm{putNonmono1}}\ (s, v))$。若 $v = \Omega$，这显然成立，因此我们只考虑 $v = ()$ 的情况。在这种情况下，根据定义，我们有 $s' = \textsf{put}\ \ell_{\mathrm{putNonmono1}}\ (s, v) \neq \Omega$，因此 $\textsf{get}\ \ell_{\mathrm{putNonmono1}}\ s' = ()$。最后，确认 ps-稳定性。假设 $\textsf{put}\ \ell_{\mathrm{putNonmono1}}\ (s_0, v) = s$，且 $\textsf{put}\ \ell_{\mathrm{putNonmono1}}\ (s', v'') = s''$，其中 $s \leq s'$ 和 $v \leq v'' \in \mathrm{I}^{get\ \ell_{\mathrm{putNonmono1}}\ s'}$。我们将证明 $s \leq s''$。由于其 put 在第二个参数上单调，只需考虑 $v'' = v$ 的情况。当 $v = \Omega$ 时，根据定义有 $s = s'' = \Omega$。假设 $v = ()$。当 $s_0 = \Omega$ 时，根据定义有$s = s' = s'' =$ 是。当 $s_0 \neq \Omega$ 时，根据定义有 $s = s' = s'' = s_0$。

C 一个不满足 WPutGet 的良态透镜

令 $\mathbf{1}_\Omega$ 为由 $\mathbf{1} = \{()\}$ 得到的自由带基偏序集。考虑以下透镜 $constUnit_{\mathrm{ns}} \in \mathrm{Lens}^{\leq}\ \mathbf{1}_\Omega\ \mathbf{1}_\Omega$。

$$get\ constUnit_{\mathrm{ns}}\ _ = ()\qquad put\ constUnit_{\mathrm{ns}}\ (s, v) = \begin{cases} s & \text{if } v = () \\ \Omega & \text{if } v = \Omega \end{cases}$$

该透镜是良态的：它满足 ps-可接受性、ps-一致性和 ps-稳定性。我们只证明 ps-稳定性，因为另外两项更直接。假设 $\textsf{put}\ \textsf{constUnit}_{\mathrm{ns}}\ (s_0, v) = s$, $s \leq s'$, $v \leq v'' \in \mathrm{I}^{get\ constUnit_{\mathrm{ns}}\ s'}$, 且 $\textsf{put}\ \textsf{constUnit}_{\mathrm{ns}}\ (s', v'') = s''$。注意，无论$s'$如何，$\textsf{get}\ \textsf{constUnit}_{\mathrm{ns}}\ s' = ()$。我们将通过对 v 进行分情况讨论来证明 $s \leq s''$。考虑情形 $v = ()$。在这种情况下，我们有 $s = s_0$。此外，$v = () \leq v'' \in \mathrm{I}^{()}$ 蕴含 $v'' = ()$。于是，根据定义，我们有$s'' = \textsf{put}\ \textsf{constUnit}_{\mathrm{ns}}\ (s, v'') = s'$。因此，根据假设 $s \leq s'$，我们有 $s \leq s'' = s'$。考虑情形 $v = \Omega$。在这种情况下，我们有 $s = \Omega$。由于 Ω 是 $\mathbf{1}_\Omega$ 中的最小元，无论 s'' 如何，我们都有 $s = \Omega \leq s''$。

相反，$\textsf{constUnit}_{\mathrm{ns}}$ 不满足WPutGet。我们有$\textsf{put constUnit}_{\mathrm{ns}}((), \Omega) = \Omega$，但我们也有$\textsf{put constUnit}_{\mathrm{ns}}((), \textsf{get constUnit}_{\mathrm{ns}}\ \Omega) = \textsf{put constUnit}_{\mathrm{ns}}((), ()) = () \neq \Omega$。