arXiv:2603.14538v1 [quant-ph] 15 Mar 2026

量子程序的可逆生命周期语义

Simone Faro[1[0000−0001−5937−5796]]、Francesco Pio Marino[1,2[0000−0003−4722−9542]]和 Gabriele Messina[1[0009−0004−9926−3691]]

[1] 卡塔尼亚大学，卡塔尼亚，意大利[2] 诺曼底鲁昂大学，法国鲁昂

摘要。 可逆计算要求显式撤销中间数据，而非将其丢弃。在量子编程中，这一原则体现为反计算，通常被视为一种技术性清理机制。本文则将反计算作为语义基础来呈现。我们在 Qutes 语言中引入《作用域界定的活性引导反计算》的形式模型：词法作用域界定变量生命周期，静态活跃性和纠缠分析则确定最早的安全回收点。我们定义了语义生命周期和恢复不变量，以确保临时量子信息一旦在语义上变得无关，便会消失。我们证明了嵌套作用域下的组合正确性，并表明提前回收能够通过避免关键路径开销来降低电路深度，还能通过规范的辅助量子比特复用限制活跃量子比特的峰值数量。最后，我们表明参数传递语义源自同一生命周期规范，其中值传递和引用传递对应不同的生命周期边界；同时，我们刻画了必须限制自动反计算的条件（不可逆性、持久纠缠和别名）。

关键词： 可逆计算 · 自动反计算 · 量子编程语言 · 参数传递语义。

1 引言

可逆计算揭示了信息无法在不付出物理代价的情况下被擦除，这一原理由 Landauer 及后续奠基性研究形式化确立[7,1,8,13]。量子电路在门级上是酉的，但量子程序常常通过辅助量子比特分配、算术子程序和结构化控制流引入临时数据。除非明确撤销这些操作，否则此类临时数据可能仍与活跃变量纠缠，从而妨碍干涉，并影响振幅放大等算法原语的效果 [6]。

反计算会将临时量子比特恢复到其初始状态（通常为 $|0\rangle$），从而消除残留纠缠并保持可逆性。在低级量子电路中，这通过显式应用逆向片段来实现。然而，在高级程序中，手动求逆容易出错，也缺乏模块化：逆运算可能分散在不同抽象层中，或与控制流交错。

有几种方法可解决这一问题。Silq 通过类型纪律 [2] 强制执行安全反计算。Unqomp 和 Reqomp 则在编译器层面合成清理电路并优化空间使用 [10,11]。模块化综合技术进一步支持组合式反计算 [14]。尽管这些方法各有不同，但它们大多将反计算视为在前向计算指定完成后再应用的一种变换。

我们采用了不同的视角。在这项工作中，以及在高层量子编程语言 *Qutes* [3], 中的具体实现中，反计算由语义生命周期决定。我们提出了作用域界定的活性引导反计算，这是一种以词法作用域界定变量生命周期，并通过静态活跃性和纠缠分析确定最早语义安全回收点的规则。临时量子数据必须满足恢复不变量：一旦变量在语义上变得无关，就必须将其恢复并释放。

在 Qutes 中，生命周期边界是语言的一等结构元素，因此能够在量子电路综合最终确定之前，对所有权、别名和纠缠进行静态推理。这一模型在量子电路和语言层面都带来了结构性影响。提前回收可在嵌套作用域中组合进行，可能避免不必要的关键路径累积，并通过有序复用辅助量子比特来限制峰值活跃宽度。在语言层面，参数传递语义由生命周期边界决定：生命周期在函数退出时结束的参数满足恢复不变量（观察意义上的值传递），而显式延长生命周期则会产生引用传递行为。我们进一步刻画了限制自动回收的确切条件，即不可逆性、持久纠缠和别名。

全文中，我们假设读者熟悉量子计算的标准电路模型。我们采用单位成本的一量子比特门和二量子比特门，并以电路深度和规模衡量复杂度 [9]。由于篇幅所限，本文不介绍量子计算的基本概念。在此框架下，我们证明，Qutes 中实现的基于生命周期的反计算并非后处理优化，而是一项组合式语义原则，它统一了可逆推理、资源管理和高层量子编程。

2 动机、相关工作与 Qutes 概述

辅助量子比特在非平凡量子程序中不可或缺。它们会出现在算术例程、结构化控制流和中间数据存储中。如果未能妥善还原，辅助量子比特可能仍与活动输出纠缠，从而改变可观察行为，并妨碍振幅放大等基于干涉的原语 [6]。因此，临时量子数据的管理不仅是实现层面的问题，也是可逆计算的一项语义要求。

[3] *Qutes* 是一种高级量子编程语言，旨在支持结构化控制流、具备所有权感知能力的量子数据，以及生命周期引导的反计算。文档和示例请访问 https://qutes-lang.org/。

表1. 不同语言和框架中的反计算机制比较。

系统	自动	作用域感知	基于生命周期	提前回收
Silq [2]	类型强制	部分	无显式生命周期模型	无
Q# [12]	伴随块	No	No	No
Quipper [5]	伴随生成	No	No	No
Unqomp [10]	编译器综合	No	No	有限
Reqomp [11]	空间感知综合	否	No	有限
Qutes（本文工作）	是	Yes	Yes	Yes

在低级量子电路描述中，反计算通过显式附加逆向片段来实现。然而，在更高的抽象层次上，逆运算可能分散在控制结构和模块边界之中。这促使人们在语言级和编译器级支持自动清理。

现有多种方法。Silq 通过类型系统强制执行安全反计算，以确保临时值得到恢复 [2]。Q# 和 Quipper 提供显式的伴随生成机制以及结构化的计算–使用–反计算模式 [12,5]。Unqomp 和 Reqomp 等基于编译器的技术，会在构造出正向电路后综合逆电路并优化空间使用 [10,11]。较新的模块化综合方法支持在电路级进行组合式清理 [14]。

尽管各不相同，这些系统大多将反计算视为：（i）一种正确性保障机制；（ii）一种应用于数据流已确定的计算的变换。

与此不同，Qutes 的设计将变量生命周期和所有权显式地置于语言级别 [4,3]。它是一种基于结构化控制流、词法作用域和面向数组的量子数据的高级量子编程语言。临时寄存器在语法区域内引入，这些区域的边界界定了其预期生命周期。这些结构信息使得我们能够在电路综合最终确定之前，对活跃性和纠缠进行静态推理。

因此，关键区别在于语义层面，而非纯粹的操作层面：在 Qutes 中，反计算由基于作用域和活跃性的生命周期分析所支配，而不是依赖显式的伴随块或事后进行电路重写。

表1总结了代表性系统与 Qutes 在反计算支持方面的主要差异。现有系统要么依赖显式的伴随结构，要么在构建正向量子电路后合成清理片段。Qutes 的不同之处在于，生命周期边界是语言中的一等结构元素。这使得在嵌套作用域内能够根据活跃性进行回收，从而有望同时减少关键路径累积和峰值活跃宽度。

除了量子特定场景之外，所提出的生命周期规范也可视为对经典可逆计算的一种资源感知细化。Bennett式清理会在正向阶段之后恢复全局状态，而生命周期引导的回收则会在语义边界强制进行局部恢复。从这个意义上说，

这种方法可解读为：为可逆资源管理引入一种结构化、类似区域的规范。它遵循逻辑可逆性的基本原则，同时针对量子纠缠和所有权约束进行了调整。本文余下部分将形式化这一基于生命周期的规范，并研究其电路级和语义后果。

3 Qutes 中的自动反计算

本节将<样式 id='1'>作用域界定的活性引导反计算</样式>形式化。我们引入一种基于依赖关系的程序模型，并为量子变量定义活跃性和语义生命周期。这些概念决定了何时可以安全地恢复并释放临时寄存器，并使我们能够分析提前回收对电路深度和宽度的影响。我们进一步研究嵌套作用域下的组合性。

作为运行示例（图1），考虑一个名为 compute 的函数，其输入为 x_1<样式 id='2'>，</样式> x_2，输出为 y_1<样式 id='6'>，</样式> y_2，并使用临时变量 t_1<样式 id='10'>，.....，</样式> t_4。在电路级，抽象的 Qutes 过程 f<样式 id='14'>，</样式> g<样式 id='17'>，</样式> h<样式 id='20'>，</样式> k 对应于酉块 U_f<样式 id='24'>，</样式> U_g<样式 id='27'>，</样式> U_h<样式 id='30'>，</样式> U_k。第一段从 x_1 准备（t_1<样式 id='34'>，</样式> t_2），并将它们的贡献传递给 y_1；第二段从（x_1<样式 id='44'>，</样式> x_2）准备（t_3<样式 id='40'>，</样式> t_4），并将它们的贡献传递给 y_2，随后在 y_2上执行 k 。假设输出初始化为 $|0\rangle$，该函数计算出

$$y_1 = f(x_1) \oplus g(x_1), \qquad y_2 = k\big(h(x_1) \oplus h(x_2)\big).$$

该示例还展示了经典的先计算后反计算策略，即在前向计算之后附加伴随块。相比之下，我们的分析将说明，回收可以由语义生命周期驱动，而非依赖全局电路终止。该示例虽然刻意保持简洁，却涵盖了依赖链、纠缠传播，以及其最后一次语法使用与最后一次语义影响并不一致的临时变量。

3.1 生命周期引导的反计算的形式化模型

我们通过酉分段之间的依赖关系和纠缠关系，对活跃性与语义生命周期进行形式化定义。

将编译后的 Qutes 程序表示为一系列酉块

$$C = (U_1, U_2, \ldots, U_n),$$

其中每个 U_i 都作用于一组量子变量。程序点用 $1 \le i \le n$ 编号，其中点 i 表示紧接在 U_i 之后的状态。

我们的形式化推导抽象掉表层语法，直接在编译后的酉序列层面定义生命周期语义。应将这种抽象理解为高级语言与其电路级实现之间的一种语义契约：假定任何 Qutes 程序都存在一种酉分解，且该分解符合词法作用域和所有权约束。因此，下文确立的结果刻画了编译器必须强制执行的语义规范，而不依赖于具体的语法结构。

```
量子比特[]compute(量子比特 x1, 量子比特 x2, 量子比特 y1, 量子比特 y2) {
p0.   量子比特 t1,t2 , t3 , t4 ;
p1.   f(x1,t1);
p2.   g(x1,t2);
p3.   受控非门 t1,t2;
p4.   受控非门 t2,y1;
p5.   h(x1,t3);
p6.   h(x2,t4);
p7.   受控非门 t3,y2;
p8.   受控非门 t4,y2;
p9.   k(y2);
p10.  return [y1 , y2];
}
```

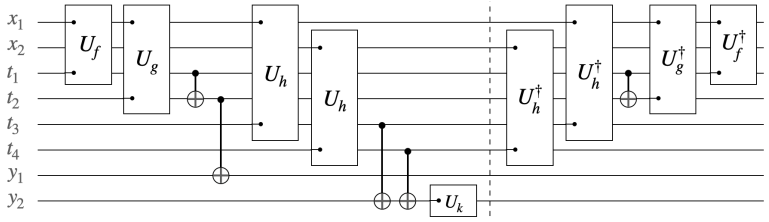


图1. 运行示例的电路级实现。前向计算（左侧）实现酉块 U_f、 U_g、 U_h 和 U_k，而最右侧部分展示了经典先计算后反计算策略，即在电路末尾附加伴随块。

定义 1（数据依赖图）。令 $C = (U_1, \ldots, U_n)$。数据依赖图 $G_D = (V, E_D)$ 定义如下：

– $V = \{1, \ldots, n\}$ 由程序点组成。– $(i \to j) \in E_D$ ，其中 $i < j$ 当且仅当 U_j 通过共享的量子变量依赖于 U_i 。

特别是，如果某个变量在 U_i 中被写入，之后又被 U_j 读取或修改，那么$(i \to j) \in E_D$。

图 G_D 描述了函数式数据流及其引发的顺序约束。在最优调度下，电路深度等于 G_D 中最长有向路径的长度。

在运行示例中（图1），相应的依赖图（左侧的图2）明确展示了临时寄存器如何通过受控相互作用将影响传递到输出。具体而言，多量子比特门引入的依赖关系将从临时变量到活动输出的影响链延伸开来。

定义 2（纠缠图）。在程序点 p处，纠缠图 G_E（p） = （Q、 E_E（p）） 是一个无向图，其中：

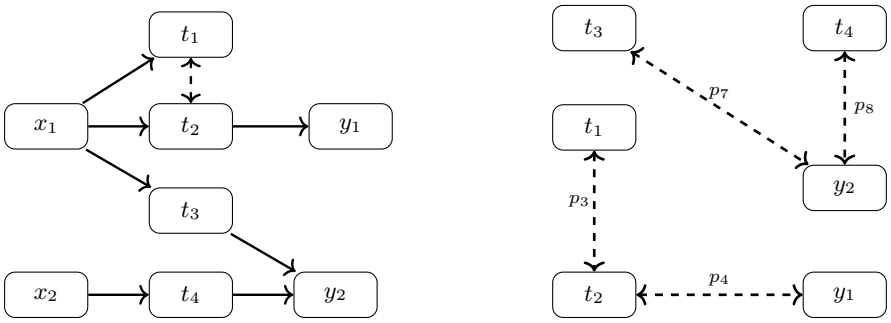


图2. （左）图1中示例的数据依赖图。实线箭头表示函数式数据依赖，虚线双向边表示纠缠诱导依赖。（右）该示例的纠缠图：虚线连接可能通过受控操作发生纠缠的变量。

– Q 是量子变量的集合；– （q_i、 q_j） $\in E_E$（p）表示此前的酉操作可能已在 q_i 与 q_j之间诱导出不可分离关联。

对于编译， $G_E(p)$通过跟踪多量子比特门进行保守近似：每当一个门同时作用于两个变量时，就在它们之间引入一条无向边。

图2（右）中的运行示例纠缠图展示了受控操作如何引入关联，而这些关联可能使临时变量的语义影响超出其最后一次语法使用。纠缠关系将用于保守地传播活跃性。

定义 3（活跃性）。如果存在一个未来操作 U_j，且 $j > p$，那么量子变量 v 在程序点 p 处是存活的：

1. v 直接参与 U_j；或者 *2.* 存在一个变量 w，使得（v、 w） $\in E_E$（p），并且 w 在 p处处于存活状态。

我们将该谓词记为live（v、 p）。

条件（2）通过纠缠传播活跃性，确保只要一个变量与存活数据相关联，该变量就会保持存活。

在图1的示例中，活跃性关系会沿着数据依赖和纠缠边传播。在 p_3 之后，CX(t_1， t_2) 门使 t_1 和 t_2 相互依赖，而后续位于 p_4 的 CX(t_2， y_1) 门则将这一对变量与活动输出 y_1 相连。由于 y_1 一直存活到计算结束，活跃性会保守地回溯传播至 t_2 和 t_1。类似地， t_3 和 t_4 在 p_5 和 p_6 处变为存活，并在与位于 p_7 和 p_8 的 y_2 交互后仍保持存活，因为 y_2 在 p_9 处被使用。这说明，纠缠诱导依赖如何使临时变量的活跃区间延伸至其最后一次语法使用之后。

定义 4（语义生命周期）。设 v 是在词法作用域 S 内声明的量子变量。v 的语义生命周期 是指最大区间

$$[p_{\min}, p_{\max}]$$

使得对于每个程序点 $p \in [p_{\min}, p_{\max}]$，存活$(v, p)$ 均成立，其中

$$p_{\min} = \min\{\, p \mid \mathrm{live}(v, p) \,\}, \qquad p_{\max} = \max\{\, p \mid \mathrm{live}(v, p) \,\}.$$

v 的词法作用域为 $p_{\max}$ 提供了语法上的上界和可能分配范围的下界，但语义生命周期可能在声明之后才开始，并在作用域结束之前终止。

此生命周期是根据基于纠缠的活性关系计算得出的，因此构成了对语义相关性的保守近似。

请注意，在整个形式化推导过程中，活跃性和语义生命周期都是针对量子资源而非语法标识符定义的。如果多个程序变量别名指向同一个底层量子寄存器，那么在活跃性分析中，它们会被视为单一资源。形式化地说，如果 v_1、...、v_k 指向同一个物理寄存器 R，那么

$$\mathrm{live}(R, p) \text{ holds iff } \exists i \ \mathrm{live}(v_i, p),$$

而 R 的语义生命周期是其所有别名生命周期的并集。因此，只有当资源本身不再存活时，才允许安全回收。

定义 5（输出可隔离子电路）。令 U 为程序点 p_a 与 p_b 之间子计算所对应的幺正变换。令 L 为 p_b 处的活跃输出变量集合。如果存在一种分解，使得该子电路相对于临时变量集合 T 被称为输出可隔离，

$$U = U_L \circ V_T$$

则：

– U_L 仅作用于 L 中的变量，– V_T 仅作用于 T 中的变量，– 并且 V_T 可逆，且不会改变 L 的约化态。

输出可隔离性是一种语义条件，而非纯粹的语法条件。一般而言，要确定一个酉算子是否允许这样的分解，可能需要对纠缠结构进行全局推理。实际上，Qutes 中实现的静态分析提供了一种保守近似：只要无法证明输出可隔离性，就会禁用自动回收。因此，以下形式化结果将输出可隔离性视为一种语义属性，同时承认其静态检测必然不完备。

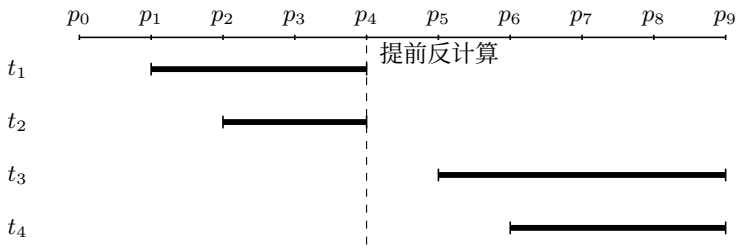


图3. 图1中示例在提前反计算下的语义生命周期。基于纠缠的活性分析保守地将 t_1 和 t_2 的存活时间延长至程序结束，但 p_4 之前的子电路是输出可隔离的；因此， t_1 和 t_2 可在 p_4 之后安全回收，而 t_3 和 t_4 则因与 y_2 相互作用而一直存活至 p_9 。

备注。 与存活变量纠缠意味着变量处于活跃状态，因此会保守地延长语义生命周期。然而，纠缠本身并不一定会妨碍安全反计算。如果某个子电路具有输出可隔离性，即使静态活跃性分析判定相关临时变量处于纠缠状态，这些变量仍可能满足提前回收的条件。因此，应将基于纠缠的生命周期理解为对语义不可逆性的一种可靠但不完备的近似。

在图1的示例中，基于纠缠的活跃性分析得出一种保守的语义生命周期： t_1 在 p_1 时变为存活；在 p_3 时与 CX(t_1， t_2) 以及在 p_4 时与 CX(t_2， y_1) 相互作用后，由于与活动输出 y_1 发生纠缠传播， t_1 会一直存活到计算结束。同样， t_2 在 p_2 时变为存活，并保守地一直存活至结束。变量 t_3 和 t_4 分别在 p_5 和 p_6 时变为存活；由于它们通过 p_7 和 p_8 与活动输出 y_2 纠缠，并且 y_2 随后在 p_9 被消耗，因此它们会一直存活至结束。

然而，在 p_4 准备 (t_1， t_2) 的子计算具备输出可隔离性之后：涉及输出 y_1 的唯一操作是 CX(t_2， y_1)，而 (t_1， t_2) 上此前的历史由 p_1–p_3 构成。因此，紧接在 p_4 之后，编译器可以插入伴随序列 CX(t_1， t_2)、 $g^{\dagger}$(x_1， t_2)、 $f^{\dagger}$(x_1， t_1)，将 t_1 和 t_2 恢复到初始状态，同时不影响 y_1 的最终值。因此，在提前反计算下， t_1和 t_2 的有效语义生命周期分别缩短为 $[p_1, p_4]$ 和 $[p_2, p_4]$，而 t_3 和 t_4 的生命周期仍分别为 $[p_5, p_9]$ 和 $[p_6, p_9]$。

定义 6（安全回收点）。如果变量 v 在程序点 p 处不再存活，且应用于 v 的任何不可逆操作所产生的影响都不会持续到 p 之后，则程序点 p 是变量 v 的安全回收点。在此类程序点，编译器可以插入应用于 v 的酉历史的伴随序列，将其恢复到初始状态。

在运行示例（图1）中，基于纠缠的活性分析保守地将 t_1 和 t_2 的生命周期延长至它们与 y_1 最后一次交互之后。然而，

在该边界处，准备 (t_1， t_2) 的代码段是输出可隔离的，因此可以在它们对 y_1 的最终语义影响结束后立即插入伴随序列。相较之下，临时变量 (t_3， t_4) 在它们对 y_2 的贡献完全确定之前无法回收，因为提前恢复它们会改变活动输出的约化态。

定义 7（关键路径）。给定数据依赖图 G_D，关键路径长度是 G_D中最长有向路径的长度。在最优调度下，这等于电路深度。

在图 2 中，关键路径对应于通向输出的最长依赖链，反映了数据和纠缠所引发依赖的最大传播范围，而与临时变量之后是否可以提前回收无关。

3.2 深度保持与缩减

作用域界定的活性引导反计算的一个重要影响涉及电路深度。在经典先计算后反计算策略中，会先前向执行临时幺正片段 U_f，并且只有在整个前向计算完成后，才会追加其伴随算子 $U_f^\dagger$。由于这两个片段都位于关键路径上， U_f 的深度贡献实际上翻了一倍。

在基于生命周期的回收中，一旦相关变量不再存活，就会在最早的安全回收点插入 $U_f^\dagger$。这允许对 $U_f^\dagger$ 进行局部调度，并在数据依赖图 G_D 允许的情况下，与后续前向计算重叠。因此，深度取决于与关键路径的交互，而非伴随块的全局句法顺序。

在运行示例（图1）中，若采用全局清理，(t_1， t_2) 的制备只有在整个计算完成后才会被求逆。采用提前回收时，伴随片段会在其对 y_1 的最后一次语义影响之后立即插入，如图4所示。在修改后的电路中，逆块不再作为尾随后缀附加，而是在生命周期边界处就地放置，并可与后续作用于不相交量子比特的计算重叠。由此得到的调度消除了临时片段沿关键路径的人为重复。

引理1（提前回收下的深度界）。设 v 是与幺正片段 U_f 相关联的临时变量，且 $U_f^\dagger$ 在其最早的安全回收点插入。令 U_{rest} 表示剩余的前向计算。在符合 G_D 的最优调度下，

$$\mathrm{depth}(C) \;\le\; \mathrm{depth}(U_f) + \max\big(\mathrm{depth}(U_f^\dagger), \mathrm{depth}(U_{\mathrm{rest}})\big).$$

证明。一旦 v不再存活， $U_f^\dagger$便只依赖于参与 U_f的量子比特。其对深度的贡献取决于它与 U_{rest}关键路径的交互。如果支撑集互不相交，这些片段便可并行执行；否则，累积过程遵循 G_D中最长的依赖链。□

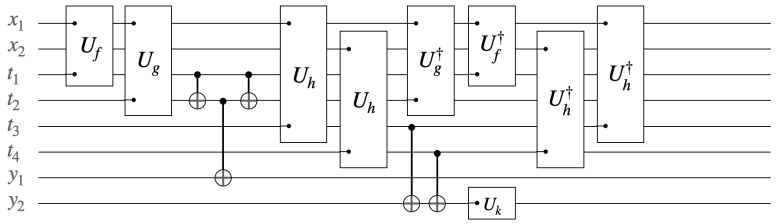


图 4. 采用提前反计算的量子电路：与 (t_1, t_2) 相关的伴随块会紧接在其最后一次语义使用之后插入，从而能够与后续计算重叠；与全局先计算后反计算相比，这会降低整体电路深度。

因此，与全局先计算后反计算相比，提前回收绝不会增加渐近深度；当 $U_f^\dagger$ 不在 U_{rest}的关键路径上时，还可能严格降低深度。

引理 2（零开销条件）。若 ***(i)*** v 在点 p之后不再存活；***(ii)*** $U_f^\dagger$ 作用于与 U_{rest}关键路径不相交的量子比特；且 ***(iii)*** $\text{depth}(U_{\text{rest}}) \geq \text{depth}(U_f^\dagger)$，则在 p处插入 $U_f^\dagger$ 不会增加电路深度。

证明。在支撑不相交的情况下，$U_f^\dagger$ 和 U_{rest} 可以并行调度。由于 U_{rest} 决定了关键路径，总深度保持不变。□

在理想情况下，生命周期引导的反计算不会增加额外深度；而全局清理策略中的伴随片段则必然会延长关键路径。

存在这样一些结构化程序：在全局先计算后反计算策略下，深度为 d 的临时片段位于关键路径上，使总深度达到 $2d + O(1)$；而通过生命周期引导插入伴随片段，可使其与独立的前向计算重叠，从而将深度降至 $d + O(1)$。因此，在存在独立子计算的情况下，基于生命周期的回收可能比全局清理实现严格的渐近改进。

3.3 辅助量子比特复用与宽度缩减

除了电路深度之外，生命周期引导的反计算还会直接影响电路宽度，即同时存活的量子变量数量。在全局先计算后反计算策略中，临时辅助量子比特通常会一直分配到计算结束，导致峰值资源使用量增加。采用提前回收时，临时变量会在其语义生命周期终止后立即恢复并释放，从而使物理量子比特能够重复使用。

定义 8（活跃集合）。在程序点 p，活跃集合为

$$L(p) = \{\, v \mid \text{live}(v, p) \,\}, \qquad W_{\max} = \max_p |L(p)|.$$

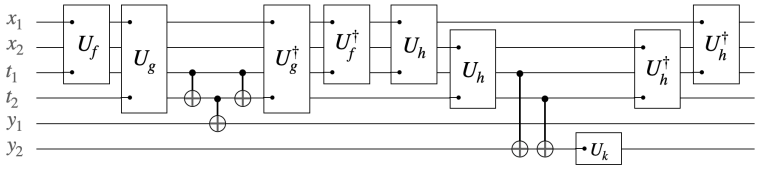


图 5. 通过提前回收获得的缩减宽度电路：临时变量 t_1 和t_2 在其最后一次语义使用后立即恢复，从而避免与 t_3 和 t_4 重叠，并减少同时存活的量子比特峰值数量。

数量 $W_{\max}$ 限定了最优回收所需的量子比特数量。

在运行示例（图 1）中，全局清理会使 t_1 和 t_2 保持分配状态，直到（t_3, t_4）处于活跃状态，从而增加峰值宽度。采用提前回收（图 5）时，与（t_1, t_2）相关联的伴随块会插入其生命周期边界，因此这些临时变量会在后续子计算达到活跃峰值之前恢复。在修改后的电路中，（t_1, t_2）和（t_3, t_4）的分配区间不再重叠，从而直接减少 $W_{\max}$。

引理 3（提前回收下的宽度界）。设 v 是一个临时变量，其语义生命周期在程序点 p结束。那么对于所有 $q > p$, $v \notin L(q)$ 均成立。因此，在 p 处插入 $U_f^\dagger$，可减少 $|L(q)|$，对于所有后续q，其中 v 原本会保持存活。

证明。 根据定义，对于 $q > p$，live(v, p) 不成立。恢复不变量确保 v 在 p 时或之前恢复到其初始状态，因此此后无需继续分配它。□

引理 4（峰值宽度缩减）。设 v_1、……、v_k 为语义生命周期两两不相交的临时变量。则存在一种调度，使所有 v_i 共用同一个物理量子比特，且 $W_{\max}$ 不随 k 增加。

证明。 在任一程序点，至多有一个 v_i 属于 $L(p)$。由于每个 v_i 都会在其生命周期结束时恢复，因此同一个物理量子比特可以按顺序重复使用。□

因此，电路宽度由语义生命周期而非词法分配决定。生命周期引导的反计算促成了可逆资源池化：临时量子比特一旦不再存活，便可重复使用。在最坏情况下，宽度与全局清理相同；而在生命周期较短且互不重叠的结构化程序中，峰值宽度会严格缩减。

3.4 提前回收的组合性

作用域界定的活性引导反计算的一个核心特性，是它在嵌套词法作用域中的组合行为。由于回收由语义生命周期而非全局结构触发，因此在

内部作用域中的提前回收不得影响外层作用域的正确性或生命周期分析。考虑如下形式的嵌套作用域

$$S_{\text{outer}}\,\{\ldots\ S_{\text{inner}}\,\{\ldots\}\ \ldots\}.$$

设 v 在 S_{inner} 中引入， w 在 S_{outer} 中引入。

引理 5（嵌套作用域独立性）。如果 p 是 S_{inner} 内针对 v 的一个安全回收点，那么在 p 处插入与 v 相关联的伴随片段，不会改变任何在 S_{inner} 外部定义的变量 w 的语义生命周期。

证明。在一个安全回收点， v 不存活，且与仍然存活的变量没有纠缠。具体而言，它无法影响 p 之外的外层作用域变量。因此，伴随片段只会影响 v 的局部量子比特，不会改变外层变量之间的依赖关系或纠缠关系。因此， w 的语义生命周期保持不变。□

因此，提前回收在嵌套作用域中保持稳定。

引理 6（生命周期单调性）。设 P 是一个程序， B 是 P 的一个子块。对于在 B 中定义的任意变量 v ，在 B 内计算出的语义生命周期包含于在 P 中计算出的语义生命周期。

证明。 活跃性是根据未来依赖和纠缠传播来定义的。将分析范围限制在 B 中，只会减少潜在的未来使用，不会引入新的使用。因此， B 中的最大活跃点不会超过 P 中的最大活跃点。□

生命周期单调性确保，在局部做出的回收决策嵌入到更大的程序中后仍然有效。

引理 7（组合式恢复）。若对于每个词法作用域 S，其中引入的所有临时变量在其语义生命周期结束时都满足恢复不变量，则整个程序满足恢复不变量。

证明。 对作用域嵌套进行结构归纳。基础情形由假设成立。对于归纳步骤，嵌套作用域独立性确保内层作用域中的回收不会影响外层变量。由于每个作用域都会在生命周期边界恢复其自身的临时变量，且不会有残留纠缠逸出，因此该不变量在每个外层级别都成立。□

组合性也延伸到电路级指标。由于回收不会引入新的跨作用域依赖，关键路径长度和活跃集界限在结构化组合下保持稳定。因此，提前回收并非全局优化阶段，而是一种组合式语义规约，与模块化编译以及对程序片段的独立推理相兼容。

4 语义后果

作用域界定的活性引导反计算的形式模型，其影响不止于电路级深度和宽度。由于生命周期边界决定了何时必须恢复临时数据，它们也由此确立了高级语言语义。具体而言，参数传递行为以及自动反计算保持可靠的条件，均直接源于语义生命周期分析和恢复不变量。

4.1 参数传递语义

在 Qutes 中，参数传递并非通过不同的运行时机制实现，而是由生命周期边界自然产生。函数对其参数的可观察影响，取决于参数的语义生命周期是否延伸至函数体之外。

默认情况下，Qutes 强制采用值传递语义。考虑

```
void f(qubit x){ ... }
```

在函数体内对 x 所做的修改均为临时修改，除非显式声明为持久化。由于 x 作为参数的语义生命周期在函数退出时结束，恢复不变量要求其约化态在调用边界处与其初始状态一致。

引理 8（涌现的值传递）。设 f 为一个函数，其参数未声明为持久化，且其函数体相对于这些参数满足恢复不变量。对于任意调用 $f(a)$，调用后a 的可观测量子态等于调用前的状态。

证明。在 f中，对 x 所做的变换必须通过返回值传播，或在其语义生命周期结束前被撤销。由于参数在函数退出后不再存活，除非它们被显式返回或标记为持久化，恢复不变量确保了在边界处完成恢复。□

这种行为并不依赖于复制量子数据，因为复制会违反不可克隆定理，而是依赖于可逆性：内部修改必须在生命周期边界处消失。

通过延长生命周期，可以获得显式的引用传递语义。使用 ref 注解后，

```
void g(ref qubit x){ ... }
```

该参数会在函数边界之后继续存活，并且在退出时不受恢复约束。因此，修改会持久化，其可观测效果与引用传递相匹配。

这种区别纯粹基于生命周期：在值传递模式下，参数的语义生命周期在函数边界处结束；在引用传递模式下，参数的语义生命周期会延伸至调用者上下文。

例如，运行示例可以写作

$$\texttt{void compute}(x_1, x_2,\ \texttt{ref}\ y_1,\ \texttt{ref}\ y_2),$$

其中， x_1， x_2 按值传递， y_1， y_2 按引用传递。

因此，Qutes 中的参数传递并非独立的执行模型，而是生命周期引导的反计算的直接体现：跨越边界的持久性必须显式声明，否则将强制恢复。

4.2 语义约束

生命周期引导的反计算受到恢复不变量以及量子计算原理所产生的语义和物理约束。这些约束精确界定了自动回收在何种情况下会违反可逆性或改变可观察行为。

第一项约束涉及不可逆操作。形式模型假设幺正演化，而测量本质上是非幺正的，并会将信息传递到经典域。

引理 9（不可逆性约束）。设 v 在程序点 p 被测量，且测量结果会影响后续计算。那么，在 p 之后， v 无法满足恢复不变量。

证明。测量会引发非幺正变换并提取经典信息。由于测量后状态取决于不再以相干方式编码的概率结果，任何幺正片段都无法在保留可观察效应的同时恢复 v 。□

第二个约束涉及有意保持的纠缠。如果输入变量与输出变量之间的相关性构成计算的预期结果，那么强制恢复输入会破坏所需的联合状态并改变语义。

别名会带来进一步的限制。活跃性必须根据物理资源而非语法标识符来计算。

引理 10（别名约束）。设 v 和 w 引用同一个量子寄存器 R。如果一个别名不再存活，而另一个仍然存活，则必须将 R 视为存活。只有当 R 本身不再存活时，才允许回收。

证明。 由于 v 和 w 表示同一物理资源，还原其中一个就需要还原 R。如果某个别名仍然存活，其状态就必须保留。因此，只有在 R 的所有别名都不再存活时，才允许进行回收。□

这些约束由语义生命周期统一起来：测量会引入不可逆效应；持久纠缠会延长生命周期；别名会阻止在标识符层级过早回收。自动反计算仅适用于演化始终保持幺正且生命周期确实终止的变量。

最后一个约束涉及跨越生命周期边界的纠缠。临时变量 x 可能与生命周期延续至边界 B之后的变量 y 发生交互。只有在能够从语义上隔离由此产生的关联时，在 B 处恢复才是合理的。

引理11（跨边界纠缠约束）。设 x 的语义生命周期在边界 B 结束，而 y 在 B 之后仍然存活。如果先前的操作使 x 与 y 发生纠缠，那么当且仅当涉及 x 的子电路相对于 y 输出可隔离时，才能在 B 恢复 x 。

证明。若满足输出可隔离性，则该子电路可分解为 $U = U_y \circ V_x$，其中 V_x 仅作用于 x ，并且可以对其求逆而不改变 y 的约化态。插入 $V_x^\dagger$ 后即可恢复 x ，同时保留 y 的可观测态。反之，若不存在这样的分解，则任何能够恢复 x 的逆操作都必然会改变联合状态，并改变 y 的约化密度矩阵，从而违反恢复不变量。□

因此，纠缠本身并不妨碍回收；只有在生命周期边界处无法隔离的关联才会阻止恢复。当输出可隔离性不成立时，无法保证自动反计算，必须显式处理恢复。因此，生命周期引导的反计算恰好受可逆推理形式化边界的限定。

5 结论

我们提出了作用域界定的活性引导反计算，将其作为 Qutes 中量子程序编译的一种语义规范。反计算不再通过追加全局清理阶段实现，而是由语义生命周期和恢复不变量决定何时必须恢复临时量子数据。生命周期在物理资源层面定义，从而确保对别名的处理正确无误。

提前回收可在嵌套作用域中进行局部判定，并具有组合性。在电路级，它能在保持最坏情况渐近复杂度不变的同时实现结构性改进：伴随片段可被调度到关键路径之外，从而降低电路深度；临时寄存器也可提前回收，从而降低峰值活跃宽度。因此，资源使用量由活跃集合而非词法分配决定。

我们通过输出可隔离性刻画了自动回收的语义边界：纠缠本身并不妨碍恢复，但无法在语义上隔离的相关性则会妨碍恢复。若无法验证可隔离性，恢复保证便不再适用。参数传递语义也遵循同一生命周期规范，其中值传递和引用传递对应不同的恢复策略。

因此，在 Qutes 中，反计算被视为一种结构化语义原则，它整合了生命周期分析、纠缠结构和可逆资源管理。未来的工作包括针对可隔离性的改进分析，以及过程间生命周期验证。

参考文献

1. Charles H. Bennett. 计算的逻辑可逆性。《*IBM*研究与开发杂志》，17(6):525–532，1973。2. Benjamin Bichsel, Daniel Zhan, Goran Sutter, and Martin Vechev. Silq：一种支持安全反计算且具有直观语义的高级量子语言。载于第*41*届*ACM SIGPLAN*编程语言设计与实现会议（*PLDI*）论文集，2020。3. Simone Faro, Francesco Pio Marino, and Gabriele Messina. 扩展 Qutes：一种实用的量子计算高级语言。《计算机杂志》，第 bxaf133 页，2025年11月。4. Simone Faro, Francesco Pio Marino, and Gabriele Messina. Qutes：一种用于简化量子计算的高级量子编程语言。载于第*34*届高性能并行与分布式计算国际研讨会（*HPDC 2025*）论文集，美国印第安纳州圣母镇圣母大学，第47:1–47:9页。ACM，2025。5. Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. Quipper：一种可扩展的量子编程语言。载于第*34*届*ACM SIGPLAN*编程语言设计与实现会议论文集，2013。6. Lov K Grover. 一种用于数据库搜索的快速量子力学算法。载于第二十八届*ACM*计算理论研讨会论文集，第212–219页，1996。7. Rolf Landauer. 计算过程中的不可逆性与热产生。《*IBM*研究与开发杂志》，5(3):183–191，1961。8. Yves Lecerf. 可逆图灵机。递归不可解性：方程 $u = \theta^n u$ 在 $n \in \mathbb{N}$ 中，其中 θ 是“编码同构”。法国科学院会议纪要，257:2597–2600，1963。9. Michael A. Nielsen and Isaac L. Chuang.《量子计算与量子信息》。剑桥大学出版社，10周年纪念版，2010。10. Anouk Paradis, Benjamin Bichsel, Samuel Steffen, and Martin Vechev. Unqomp：综合量子电路中的反计算。载于第*42*届*ACM SIGPLAN*编程语言设计与实现国际会议（*PLDI '21*）论文集，第222–236页。计算机协会，2021。11. Anouk Paradis, Benjamin Bichsel, and Martin Vechev. Reqomp：面向量子电路的空间受限反计算。《*Quantum*》，8:1258，2024年2月。12. Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#：借助高级领域特定语言实现可扩展量子计算与开发。载于*2018*年真实世界领域特定语言研讨会论文集，2018。13. Tommaso Toffoli. 可逆计算。技术备忘录 MIT-LCS-TM-151，麻省理工学院计算机科学实验室，1980。14. Hristo Venev, Timon Gehr, Dimitar Dimitrov, and Martin Vechev. 高效量子反计算的模块化综合。arXiv 预印本，2024。