唯一性即分离

Liam O'Connor 爱丁堡大学 苏格兰爱丁堡
l.oconnor@ed.ac.uk

Pilar Selene Linares Arévalo 墨尔本大学 澳大利亚墨尔本
plinaresarev@student.unimelb.edu.au

Christine Rizkallah 墨尔本大学 澳大利亚墨尔本
christine.rizkallah@unimelb.edu.au

arXiv:2602.06386v1 [cs.PL] 6 Feb 2026

摘要

值独立性对于大规模推理软件系统极为有益。这些益处也延伸到了形式化验证领域。在证明助手中，对程序进行代数推理十分简单；而对于不受约束的可变性，则需要复杂得多的技术，例如分离逻辑，在这种逻辑中，必须通过手动证明来确立有关内存安全、别名和状态变化的不变量。

唯一性类型系统允许将程序编译为使用变更操作以提高效率的代码，同时保留具有值独立性语义，以便进行推理。然而，这些类型系统的限制往往过于严苛，不适用于实际软件。因此，大多数唯一性类型系统都包含某种"逃生口"：它会失去值独立性带来的推理优势，但同时解除唯一性类型的限制。要对这种具有混合保证的系统进行形式化验证，必须用命令式、可变语义来表达唯一性类型所保证的值独立性。换言之，我们应当将值独立性表述为分离逻辑中的断言。

1 引言

唯一性类型 [8]允许我们在推理时将程序视为其中所有数据结构均不可变，并享有由此带来的所有益处，同时在实际实现中对可变数据结构执行高效的破坏性更新。为此，系统通过静态方式排除所有可能观察到不可变解释与可变解释之间差异的程序，具体做法是要求每个可变值在任一时刻都只有一个存活的引用。这称为**唯一性条件**，它保证了必要的无别名性质，从而确保不可变语义与可变语义一致。Cogent 语言 [6]利用这些推理优势生成易于推理的纯函数式规范，并通过形式化证明将其与使用破坏性更新的高效 C代码关联起来。

唯一性条件是一项非常简单的限制，但在实际编写程序时却可能造成相当大的负担。例如，简单的唯一性类型系统会禁止同时将一个数组及其某个元素的引用传递给函数，因为这会引入别名，即使我们只读取这些引用，且不涉及任何修改也是如此。具有复杂布局、涉及许多共享引用和别名引用的复杂数据结构，根本无法表达。

因此，大多数唯一性类型系统都包含某种"逃生通道"，允许暂时解除类型系统施加的限制。例如，在 Rust 中，[5]，不安全代码块中的代码可能违反唯一性条件。Cogent 同样提供了外部函数接口（FFI），允许将程序的部分内容用不安全的命令式 C 语言编写。这些 C 组件能够操作不透明类型，而这些类型在系统中纯函数式的 Cogent 组件里是抽象的。于是，Cogent 认证编译器的精化定理会包含这样的假设：C 组件是安全的，并且不会违反系统中 Cogent 部分的唯一性条件。要验证混合的 Cogent-C 系统，就需要通过手动证明来证实这些假设对 C 代码成立，具体如 Cheung等人 [3]所述。

本文阐述了 Cogent 对引用和可变存储施加的义务[1]，，并阐明了我们为何相信分离逻辑 [7] 能够提供恰当的语言来表达这些义务，以及恰当的证明演算来履行这些义务。

2 强制唯一性

Cogent 的静态类型系统相当复杂，我们在此不再赘述[2]。就本文而言，我们只关注它所保证的动态属性。Cogent 的类型保持定理表明，一个类型为 $\tau \to \rho$的函数，在给定一个类型为 τ的值 v时，确实会返回一个类型为 ρ的值。此外，Cogent 将值的动态类型关系 $v :\ \tau\ p$扩展为还包含可从该值访问的一组指针 p，称为其 **堆足迹**[3]。例如，元组值的类型规则如下：

$$\frac{x : \tau_1\ \langle p \rangle \quad y : \tau_2\ \langle q \rangle \quad p \cap q = \emptyset}{(x, y) : \tau_1 \times \tau_2\ \langle p \cup q \rangle}$$

观察这些指针集合如何用于强制结构中不存在内部别名（前提 $p \cap q = \emptyset$）。这是因为元组对 Cogent 而言并非抽象的，因此任何内部别名都可能导致违反唯一性条件。对于对 Cogent 而言抽象的、由 C 实现的外部数据结构，可以允许内部别名。

该类型关系还提供了精确陈述以下条件所需的信息：在这些条件下，C 程序不会干扰 Cogent 所享有的内存安全或值独立性保证：令 σ 表示一个存储，即从指针 ℓ 到值 v的偏映射。令 f: $\tau \to \rho$ 为用 C 实现并导入 Cogent 的函数。若以输入值 $v :\ \tau\ p$和输入存储 σ对 f求值，则返回值 $f\ v :\ \rho\ p'$和输出存储 σ' 必须对所有指针 ℓ 满足以下三个性质：

无泄漏性 $\ell \in p \wedge \ell \notin p' \Rightarrow \ell \notin \mathrm{dom}(\sigma')$，即任何未被返回的输入引用都已释放。

新鲜分配 $\ell \notin p \wedge \ell \in p' \Rightarrow \ell \notin \mathrm{dom}(\sigma)$，也就是说，每个不在输入中的新输出引用，都是在之前空闲的空间中分配的。

VIMPL，日本东京
2026。

[1]还有其他义务。完整列表请参见 Cheung等人 [3] 。[2]完整描述请参见 O'Connor 等人 [6] 。[3]实际的内存占用范围稍微复杂一些，还需要考虑借用，此外还包括一组可共享的只读指针。

惯性 $\ell \notin p \wedge \ell \notin p' \Rightarrow \sigma(\ell) = \sigma'(\ell)$，也就是说，既不在函数输入中也不在函数输出中的每个引用，都没有以任何方式被修改。

假设满足这三个性质，即使带有唯一性类型的程序依赖于不安全的命令式 C代码，也可以证明其两种语义解释是等价的。这三个条件称为框架条件，其名称源自知识表示领域中的框架问题。

3 分离逻辑

上述框架问题是对有状态进程进行形式化时经常出现的一个常见问题。具体而言，它指的是局部推理的困难。例如，典型的命令式程序适合采用公理语义进行验证，最显而易见的例子便是霍尔逻辑 [4],，它为如下形式的判断提供了一种证明演算 $\mu \models \{\phi\}P\{\psi\}$。这表示，假设初始状态 μ（将变量映射到值）满足断言 ϕ，那么运行程序 P 于 μ之后得到的结果状态满足 ψ。基于霍尔逻辑的验证框架适用于简单程序，但验证操作堆内存的程序则既繁琐又困难。必须始终维护若干不变量，以说明引用不会产生别名、引用指向空闲空间，或引用指向有效值。这是因为堆被视为一个单一的整体结构。因此，每当堆的任何部分被更新时，都必须重新确立关于堆的每个不变量，即使它与此次更改无关——这就是框架问题。

Cogent 的框架条件规定，任何函数（包括用 C 实现的函数）都不会影响堆中任何不在其获准修改范围内的部分（其权限由它收到的引用决定），从而确保程序其他部分的不变量得以保持。对于 Cogent 代码自动生成的证明而言，这样表述框架条件是可行的；但在验证系统的 C组件时，直接用堆和指针来表述这些证明义务仍然繁琐且困难 [3]。当不变量必须先被破坏、之后才能重新建立时，这一点尤为明显。

为了使这一过程更加简洁，我们转而采用 Reynolds 的分离逻辑 [7]。分离逻辑是霍尔逻辑的一种变体，专为支持使用引用和别名的编程而设计。除了霍尔逻辑中的状态 μ，我们还有可变存储 σ，以及以下额外的断言：

- 一个特殊断言 **emp**，表示存储为空，即 $\mu, \sigma \models \mathbf{emp}$ 当且仅当 $\mathrm{dom}(\sigma) = \emptyset$。
- 一个二元运算符 $\mapsto$：$\ell \times v$，它表示存储恰好在一个位置上有定义，即 $\mu, \sigma \models \ell \mapsto v$ 当且仅当 $\mathrm{dom}(\sigma) = \{\ell\} \wedge \sigma(\ell) = v$。
- 一个分离合取连接词 $\phi * \psi$，它表示存储 σ可以拆分为两个互不相交的部分 σ_1和 σ_2，其中 $\mu, \sigma_1 \models \phi$且 $\mu, \sigma_2 \models \psi$。
- 一个分离蕴含连接词 $\phi \mathrel{-\!\!*} \psi$，它表示用一个满足 ϕ的互不相交部分扩展存储后，得到的存储满足 ψ。

至关重要的是，分离逻辑包含框架规则，这是它对框架问题提出的解决方案，其中可以添加一个无关的断言 ϕ_r 将其加入给定程序的前置条件和后置条件中的分离合取：

$$\frac{\{\phi\}\ P\ \{\psi\}}{\{\phi * \phi_r\}\ P\ \{\psi * \phi_r\}}$$

这使我们能够进行与之前所期望的几乎相同的局部推理：可以验证程序 P 在满足 ϕ、但除此之外不含其他值的存储上能够正常工作。随后便可将该程序自由地用于更大的状态，并且我们可以根据框架规则自动得知，任何无关的状态部分都不会影响程序，也不会受到程序的影响 P。

分离逻辑使表达我们的框架条件变得简单得多。给定一个程序 P，其输入指针集为 p，输出指针集为 p'，我们可以将全部三个条件表达为单个三元组：

$$\left\{ *_{\ell \in p}\ \exists v.\ \ell \mapsto v \right\} P \left\{ *_{\ell \in p'}\ \exists v.\ \ell \mapsto v \right\}$$

我们概述一个证明，说明这蕴含了上面列出的框架条件。假设输入存储为 σ。将 σ 分割成不相交的存储 σ_1 和 σ_2，使得 $\sigma_1 \models *_{\ell \in p}\ \exists v.\ \ell \mapsto v$（$*$）。令使用 σ_1 运行 P 后的输出存储为 σ_1'。注意，根据上面的三元组，我们有 $\sigma_1' \models *_{\ell \in p'}\ \exists v.\ \ell \mapsto v$（$**$）。利用框架规则，我们知道使用完整存储 σ 运行 P 的输出为 $\sigma' = \sigma_1' \cup \sigma_2$，其中 $\mathrm{dom}(\sigma_1') \cap \mathrm{dom}(\sigma_2) = \emptyset$。

无泄漏性 对于任意位置 ℓ，如果 $\ell \in p$ 但 $\ell \notin p'$，则我们必须证明 $\ell \notin \mathrm{dom}(\sigma')$。由于 $\ell \in p$，根据（$*$）我们知道 $\ell \in \mathrm{dom}(\sigma_1)$，并且由于它们不相交，$\ell \notin \mathrm{dom}(\sigma_2)$。因此，$\ell \in \mathrm{dom}(\sigma')$ 为真唯一可能的情况是 $\ell \in \mathrm{dom}(\sigma_1')$；但由于 $\mathrm{dom}(\sigma_1') = p'$ 来自（$**$），我们可以得出结论：$\ell \notin \mathrm{dom}(\sigma')$。
新鲜分配 如果 $\ell \notin p$ 但 $\ell \in p'$，则我们必须证明 $\ell \notin \mathrm{dom}(\sigma)$。根据（$**$）我们有 $p' = \mathrm{dom}(\sigma_1')$，因此 $\ell \in \mathrm{dom}(\sigma_1')$。由于它们不相交，$\ell \notin \mathrm{dom}(\sigma_2)$，所以 $\ell \in \mathrm{dom}(\sigma)$ 为真的唯一可能情况是 $\ell \in \mathrm{dom}(\sigma_1)$。但是，由于我们知道 $\mathrm{dom}(\sigma_1) = p$ 来自（$*$）以及 $\ell \notin p$，我们可以得出结论：$\ell \notin \mathrm{dom}(\sigma)$。
惯性 如果 $\ell \notin p$ 且 $\ell \notin p'$，那么根据（$*$）我们可以得出 $\ell \notin \mathrm{dom}(\sigma_1)$，并根据（$**$）得出 $\ell \notin \mathrm{dom}(\sigma_1')$。如果 $\ell \in \mathrm{dom}(\sigma_2)$，那么根据上面的框架规则，$\sigma(l) = \sigma_2(l) = \sigma'(l)$。如果 $\ell \notin \mathrm{dom}(\sigma_2)$，那么 $\ell \notin \mathrm{dom}(\sigma)$ 且 $\ell \notin \mathrm{dom}(\sigma')$。

该演示还使得能够重用现有的成功框架，以基于分离逻辑进行 C 语言验证 [1]。

4 结论

我们认为，这种优雅的框架条件表述，证明了唯一性类型与分离逻辑之间存在深刻联系。类型系统和程序逻辑都是形式化推理的工具；如果我们没有意识到它们之间的联系，就有重复发明的风险。这些联系可以成为一种未来语言的基础：它将精化类型 [2]与唯一性类型结合起来，编译为高效的命令式代码，并将基于类型的断言转换为分离逻辑中的断言，从而实现与手写底层扩展的无缝集成和验证。这样的语言能够减少现有验证工作中令人厌烦的障碍，并拓展经验证的软件开发的可能性。

引用

[1] Callum Bannister、Peter Höfner 和 Gerwin Klein。《分离逻辑中的前行与回溯》。载于 Jeremy Avigad 和 Assia Mahboubi 编，交互式定理证明，第 68–87 页，瑞士 Cham，2018 年。施普林格国际出版社。ISBN 978-3-319-94821-8。[2] Gavin M. Bierman、Andrew D. Gordon、Cătălin Hriţcu 和 David Langworthy。《使用 SMT 求解器实现语义子类型》。 *SIGPLAN* 通讯，45(9):105–116，2010 年 9 月。ISSN 0362-1340。doi: 10.1145/1932681.1863560。URL https: //doi.org/10.1145/1932681.1863560。 [3] Louis Cheung、Liam O'Connor 和 Christine Rizkallah。《克服限制：将外部函数的验证与 Cogent 组合》。载于 第 *11* 届 *ACM SIGPLAN* 认证程序与证明国际会议论文集，CPP 2022，第 13–26 页，美国纽约州纽约市，2022 年。美国计算机协会。ISBN 9781450391825。doi: 10.1145/3497775.3503686。[4] C. A. R. Hoare。《计算机程序设计的公理基础》。 *ACM* 通讯，12(10):576–580，1969 年。ISSN 0001-0782。doi: 10.1145/363235.363259。URL https: //doi.org/10.1145/363235.363259. [5] Steve Klabnik 和 Carol Nichols。 *Rust*程序设计语言。No Starch Press，美国，2018年。ISBN 1593278284。[6] Liam O'Connor、Zilin Chen、Christine Rizkallah、Vincent Jackson、Sidney Amani、Gerwin Klein、Toby Murray、Thomas Sewell 和 Gabriele Keller。Cogent：唯一性类型与认证编译。 函数式编程期刊，31 25，2021 10 1017 095679682100023 :e . doi: . /S X。[7] J.C. Reynolds。分离逻辑：一种用于共享可变数据结构的逻辑。载于第*17*届 *IEEE* 计算机科学中的逻辑年度研讨会论文集，第55–74页。IEEE，2002年。doi: 10.1109/LICS.2002.1029817。 [8] Sjaak Smetsers、Erik Barendsen、Marko van Eekelen 和 Rinus Plasmeijer。通过一种包含图的唯一性信息的类型系统，保证安全的破坏性更新。载于 Hans Jürgen Schneider 和 Hartmut Ehrig 编，计算机科学中的图变换，第358–379页，柏林，海德堡，1994年。Springer Berlin Heidelberg。ISBN 978-3-540-48333-5。doi: 10.1007/3-540-57787-4_23。