arXiv:2602.19868v1 [cs.PL] 23 Feb 2026

结合小步语义与大步语义验证循环优化

David Knothe FZI 信息技术研究中心，德国卡尔斯鲁厄

Oliver Bringmann 图宾根大学，德国图宾根 FZI 信息技术研究中心，德国卡尔斯鲁厄

摘要

经验证的编译器旨在保证编译能够保持源程序的可观察行为。虽然小步语义广泛应用于这类编译器，但对于循环优化等结构变换而言，它并不总是最方便的框架。本文提出了一种兼用小步语义与大步语义的方法：使用小步语义处理局部变换，而使用大步语义处理结构变换。

本文引入抽象行为语义，作为两种风格之间的通用接口。本文扩展了余归纳大步语义，使其能够正确处理具有有限轨迹与无限轨迹的发散，从而使其表达能力与小步语义相当。这样一来，便可将大步变换插入现有小步处理流水线的中间，同时完整保持所有顶层语义保持定理。本文在 CompCert 中通过采用大步 Cminor 实现并验证若干新的循环优化，展示了该方法的实际应用，其中包括循环外提，尤其是完全循环展开。

2012年 美国计算机协会学科分类计算理论 → 程序语义；软件及其工程 → 编译器；软件及其工程 → 软件验证

关键词和短语经验证的编译、语义保持性、循环展开、自然语义、余归纳语义、CompCert

补充材料软件： `https://github.com/knothed/CompCert-loop`

1 引言

你能信任你的编译器吗？这是Leroy等人多篇有影响力论文的开篇问题，这些论文介绍了CompCert——首个现实可用且已商业发布的经验证编译器 [10, 2, 11, 8]。经验证编译器的目标和价值显而易见：通过对其所有变换和优化（其中许多相当复杂）进行机器检查的证明，可以系统地排除这些组件中的错误。这有效弥合了程序书面源代码的正确功能与最终编译所得机器代码的正确功能之间的差距。虽然这种方法对于大多数普通应用来说可能有些过度，但对安全关键软件而言，它可能至关重要。

理想的正确性陈述可以如下所示：如果源程序 S 被编译为可执行程序 C，那么 S 的可观察行为与 C的相同。

为了对程序行为进行推理，源语言和目标语言都必须配备能够规定执行模型的形式语义。为此，通常会采用两种语义：操作语义（小步语义）描述程序逐步执行的过程（通常与硬件密切相关）；自然语义（大步语义）则从更概念化的高层视角来理解执行。因此，在推理变换的正确性时，这两种语义自然适用于不同的目的。小步语义尤其适用于仅在局部修改代码的变换（即使这类变换需要全局分析），例如指令选择或常量传播；

相比之下，大步语义更适用于针对代码结构进行操作的变换，例如内联，尤其是循环优化。在此背景下，Bertot 等人在 2004 [2] 中写道：

> “……我们设想，不在非结构化的 RTL 中间代码上执行诸如循环优化之类的优化，而是在结构化的 Cminor 源代码上执行，因为其大步语义使得重新排列计算更为容易，无需担心不匹配的中间计算状态。”

然而，CompCert 最初使用的大步语义后来已被小步语义完全取代，理由是小步语义表达能力更强，并提供了一个用于保持性证明的通用框架 [12, 13]。因此，官方的 CompCert 开发中并未包含任何循环变换。近年来，已有方法被提出，用于在小步语义上执行循环优化[15, 7, 6]。尽管这些方法功能强大且前景可期，但由于依赖解释器，它们仍然较为复杂，而且只适用于一类能够在控制流图上构成态射的变换。

本文旨在证明，大步语义不仅比小步语义更适合某些类型的结构变换，而且也能够集成到现有的小步语义流程中，同时保持关于程序行为的所有论述。利用这种方法，我们在 CompCert 中验证了若干新的循环变换。我们的方法和贡献可概括如下：

1. 我们从小步语义和大步语义中抽象出程序行为，为二者建立一个最小的通用接口。我们为这种所谓的行为语义定义性质，并证明其保持性定理（第 2 节）。**2.** 我们扩展了 Leroy 的余归纳大步语义 [14]，允许发散判断中包含有限轨迹与无限轨迹，使其表达能力达到小步语义的水平（第 3 节）。**3.** 我们用大步语义实现了若干循环变换 `Cminor` [1]，包括循环切换和完全循环展开（第 5 节）。据我们所知，此前 CompCert 中尚未验证过这些变换。我们讨论了大步方法的优势与局限（第 6 节）。**4.** 我们证明了大步语义与小步语义之间的行为等价 `Cminor` （第 4 节），并将上述内容全部集成到 CompCert 中（第 2.4 节）。因此，我们完整保留了其所有顶层正确性定理。该开发成果已公开[9]。

我们的贡献并不局限于 CompCert：我们希望倡导这样一种理念：对于经验证的编译器，一般而言，**使用小步语义证明某些变换，使用大步语义证明其他变换**。我们采用的行为抽象、对 Leroy 余归纳大步语义的扩展，以及由此实现的对大步行为进行充分推理的能力，都可以应用于 CompCert 之外。不过，我们在此以 CompCert 作为我们方法的具体示例，以便加以说明。

2 程序行为

经验证的编译器的总体目标，是保证编译后的程序相对于原始程序的行为。为此，程序行为的概念旨在涵盖两个关键方面：第一，程序是终止、发散，还是因未定义行为而出错；第二，执行过程中产生的可观察 轨迹

。轨迹是执行过程中产生的（可能无限长的）外部可见事件序列，例如对 `read` 或 `printf` 的调用。它不会记录执行的每一项操作，因为这会过度限制编译器优化，例如操作重排序或控制流变更。因此，一个合理的约定是：轨迹记录对所有外部函数的调用，因为这些调用可能在外部可见，而且通常无法对其行为作出任何保证。

设 p 是一个程序。[1] 我们使用三个谓词 $\mathrm{Beh}_p(t,\ \texttt{OK})$、$\mathrm{Beh}_p(\tau,\ \infty)$ 和 $\mathrm{Beh}_p(t,\ \texttt{err})$，分别表示 p 终止、发散或出错。请注意，终止和出错都对应有限轨迹 t，而发散则会产生一条一般轨迹 τ，也就是说，它可以是有限轨迹，也可以是无限轨迹。要理解原因，请考虑一个发散的 `while true`-循环。它可能永远不发出任何事件，也可能无限次调用外部函数（即，调用次数多于任何有限次数）。

一般来说，一个程序可能有多种行为，因为外部调用可能具有非确定性。第 2.2 节将借助确定性属性对此进行更详细的描述。为了讨论行为，我们必须为程序（或编译器中的中间表示）赋予语义。为此，两种常见的语义类型是小步（或操作）语义和大步（或自然）语义。

小步语义由其执行步骤刻画。判断式 $\sigma_1 \xrightarrow{t} \sigma_2$表示从程序状态 σ_1 到 σ_2的一步，这通常对应于执行单个操作。该步骤会产生轨迹 t，它也可能是空轨迹 ε。如果程序 p的执行最终到达最终状态，则该程序终止。如果执行无限继续，p 就会发散。如果非最终状态 ($\sigma \not\to$) 无法执行任何一步，则 p 发生了错误。

大步语义不采用逐步执行，而是以递归方式定义终止和发散。例如，如果先成功执行 s_1，再成功执行 s_2，语句 $s_1 \cdot s_2$ 就会终止。正如 Leroy 所述 [14], ，大步语义可以用一个用于终止的归纳谓词和一个用于发散的余归纳谓词来定义，CompCert 也采用了这种方法。[2] 通常，大步语义缺少“执行出错”判断式。

▶ 注 1。我们将 t 记作有限轨迹，将 ω 记作无限轨迹，并将 τ 记作（一般的）轨迹。

2.1 行为语义

现在我们来定义行为语义。在这里，错误运行通过部分执行隐式地表达。之后，我们将展示如何把小步语义和大步语义都解释为行为语义。

给定程序 p，p 的行为语义由三个谓词定义：$p \Downarrow^t \sigma$（以最终状态 σ终止）、$p \Downarrow^\tau_\infty$（发散）以及 $p \Downarrow^t$（p 有一条有限轨迹 t的部分求值）。部分求值需要满足以下三个性质：

(1) $p \Downarrow^\varepsilon$成立。**(2)** 设 τ 为一条轨迹，且满足 $p \Downarrow^\tau \sigma$，$p \Downarrow^\tau_\infty$，或 $p \Downarrow^\tau$ 成立。那么，对于每个有限前缀 $t \le \tau$，都有 $p \Downarrow^t$。**(3)** 紧致性：设 ω 为一条无限轨迹，并假设 $\forall t < \omega$：$p \Downarrow^t$。那么 $p \Downarrow^\omega_\infty$ 成立。

[1] 我们总是隐式包含执行环境的初始状态，例如内存和寄存器。[2] 由于这两个谓词的性质不同，在 Rocq 中很难将它们合并为一个谓词。定义统一的大步语义有几种方法，例如 [3] 和 [5], ，这里不作讨论。

▶**定义 2**。 行为 Beh_P 的 p 定义如下：$\mathrm{Beh}_P(t, \mathit{OK}\,\sigma) :\Leftrightarrow p \Downarrow^t \sigma\mathrm{Beh}_P$
$(\tau, \infty) :\Leftrightarrow p \Downarrow^\tau_\infty \mathrm{Beh}_P(t, \mathit{err}) :\Leftrightarrow p \Downarrow^t$ 和
$\neg\left(p \Downarrow^t \sigma \ \vee\ p \Downarrow^t_\infty \ \vee\ \exists t' \neq \varepsilon,\ p \Downarrow^{t \cdot t'}\right)$。

一个重要的推论是，每次部分执行都会产生一个行为。因此，根据上述性质 (1)，每个程序至少有一个行为：

▶ **引理 3.** 如果 $p \Downarrow^t$ 那么存在一个行为 b 和一条轨迹 τ，使得 行为$_P(t \cdot \tau, b)$。

证明。 经典地[3] 并利用性质 (2)，以下情况之一成立：
1. $\exists\omega$ 无限的，使得 $\forall t' < \omega,\ p \Downarrow^{t \cdot t'}$。根据紧致性 (3) 可得 $p \Downarrow^t_\omega$。**2.** $\exists t'$ 有限，使得 $p \Downarrow^{t \cdot t'} \wedge \forall \tilde{t} \neq \varepsilon,\ \neg(p \Downarrow^{t \cdot t' \cdot \tilde{t}})$。如果 $p \Downarrow^{t \cdot t'} \sigma$ 或 $p \Downarrow^{t \cdot t'}_\infty$ 成立，则证明完毕。否则，根据定义可得 $\mathrm{Beh}_P(t \cdot t',\ \mathtt{err})$。◀

2.2 性质与保持性

确定性与接纳性

非确定性的一个典型来源是调用外部函数。比如， `read` 调用的返回值事先并不确定。如果外部调用是唯一的非确定性来源，则称该语义是确定性的。在小步语义 $\rightarrow$中，确定性定义如下 [19]：如果存在两个可能的步 $s \rightarrow^{t_1} s_1$ 和 $s \rightarrow^{t_2} s_2$，那么 $t_1 \asymp t_2$ 和 $t_1 = t_2 \Rightarrow s_1 = s_2$。这里，$t_1 \asymp t_2$ 表示两条轨迹匹配：$|t_1| = |t_2|$，而且如果它们表示一次外部调用，那么调用参数相同，但返回值可能不同。[4] 因此，在确定性语义中，只有在产生轨迹事件的步骤之后，才可能出现多个不同的执行。

可接收性是一种互补性质：$s \rightarrow^{t_1} s_1 \wedge t_1 \asymp t_2 \Rightarrow \exists s_2,\ s \rightarrow^{t_2} s_2$。CompCert 假设所有外部函数都具有确定性和可接收性。现在我们为行为语义定义类似的性质：

▶ **定义 4 (**确定性与接纳性**)**。行为语义 p 是
（**a**）确定的，当：**a.** 给定$|e_1,\ e_2| = 1,\ p \Downarrow^{t \cdot e_1} \wedge p \Downarrow^{t \cdot e_2} \Rightarrow e_1 \asymp e_2$**b.** $\mathrm{Beh}_P(t,\ b_1) \wedge \mathrm{Beh}_P(t \cdot \tau,\ b_2) \Rightarrow \tau = \varepsilon \wedge b_1 = b_2$。
（**b**）可接收的，当：给定 $|e_1,\ e_2| = 1,\ p \Downarrow^{t \cdot e_1} \wedge e_1 \asymp e_2 \Rightarrow p \Downarrow^{t \cdot e_2}$

行为保持

▶ **定义 5 (**细化**)**。行为 (τ_1, b_1) 细化 行为 (τ_2, b_2) ——记作$(\tau_1, b_1) \preccurlyeq (\tau_2, b_2)$——当且仅当 $\tau_1 = \tau_2 \wedge b_1 = b_2$，或者 $b_2 = \mathit{err} \wedge \tau_2 \leq \tau_1$ （其中 τ_2 有限）。

▶**定义 6** （保持性）。 设 Beh_P 与 $\mathrm{Beh}_{P'}$ 为两种行为语义。
1. $P \rightrightarrows P'$ （前向保持）：$\Leftrightarrow$ 对于每一个行为 $\mathit{Beh}_P(\tau, b)$，都存在一个行为 $\mathit{Beh}_{P'}(\tau', b')$，使得 $(\tau', b') \preccurlyeq (\tau, b)$。*2.* $P \leftleftarrows P'$ （反向保持）：$\Leftrightarrow$ 对于每一个行为 $\mathit{Beh}_{P'}(\tau', b')$，都存在一个行为 $\mathit{Beh}_P(\tau, b)$，使得 $(\tau', b') \preccurlyeq (\tau, b)$。

[3] 基本引理 3 以及由此得出的我们的许多定理，都是以经典方式而非构造方式证明的。然而，CompCert 中情况同样如此：一些关于行为的关键定理，例如小步模拟蕴含行为保持，是以经典方式证明的。[4] 我们还额外要求单步至多产生一个事件，即 $s \rightarrow^t s' \Rightarrow |t| \leq 1$。

3. $p \equiv p'$（等价）：$\Leftrightarrow$ 对于所有 $(\tau,\ b)$，$\mathrm{Beh}_p(\tau,\ b) \Leftrightarrow \mathrm{Beh}_{p'}(\tau,\ b)$。

如果 p 被编译（或转换）为 p'，那么 $p \leftleftarrows p'$ 是最理想的正确性属性：p' 的每一个行为要么同样是 p 的行为，要么是 p [10]某一行为的细化。特别地，编译器可以将 p 中的未定义行为替换为 p'中的某种其他行为，但不得改变直至未定义行为发生点为止的可观察轨迹。细化和三种形式的保持性都是传递的。

只需展示三个行为谓词之间的蕴含关系，就能非常自然地证明变换的前向保持性（示例见第 5 节）：

▶ **引理 7**。 $p \rightrightarrows p' \Leftrightarrow \left[(p \Downarrow^t \sigma \Rightarrow p' \Downarrow^t \sigma) \ \wedge \ (p \Downarrow^\tau_\infty \Rightarrow p' \Downarrow^\tau_\infty) \ \wedge \ (p \overset{t}{\Downarrow} \Rightarrow p' \overset{t}{\Downarrow}) \right]$。[Rocq] ◀[5]

幸运的是，其他形式的保持性都可以仅由前向保持性推导得出：

▶ **定理 8**。 设 p 是可接收的，且 p' 是确定性的。那么，$(p \rightrightarrows p') \ \Rightarrow \ (p \leftleftarrows p')$。

证明概要。 目标是证明 $\mathrm{Beh}_{p'}(\tau, b) \Rightarrow \mathrm{Beh}_p(\tau, b) \vee \exists t \leq \tau, \mathrm{Beh}_p(t, \mathtt{err})$。我们转而证明它的一个 Partial 版本：$p' \overset{t}{\Downarrow} \Rightarrow p \overset{t}{\Downarrow} \vee \exists t' < t, \mathrm{Beh}_p(t', \mathtt{err})$。利用 p'的确定性和紧致性，可以将其推广到完整行为，从而得出该定理。

我们通过对 $|t|$ 进行归纳来证明其部分版本。基例 $p \overset{\varepsilon}{\Downarrow}$ 由性质 (1) 给出。现在令 $p' \overset{t}{\Downarrow} \cdot \overset{e}{}$，其中 $|e| = 1$ 是某个事件。根据性质 (2) 和归纳假设，要么 $p \overset{t}{\Downarrow}$，要么对于某个 $t' < t$，有 $\mathrm{Beh}_p(t', \mathtt{err})$。第二种情况即得证。否则即为 $p \overset{t}{\Downarrow}$。经典地，要么 $\nexists \tilde{e}$，且 $|\tilde{e}| = 1$ 和 $p \overset{t \cdot \tilde{e}}{\Downarrow}$，此时我们得到所需的 $\mathrm{Beh}_p(t, \mathtt{err})$；要么我们有 $p \overset{t \cdot \tilde{e}}{\Downarrow}$，并且利用引理 7，得到 $p' \overset{t \cdot \tilde{e}}{\Downarrow}$。$p'$ 的确定性于是推出 $e \asymp \tilde{e}$，再由 p 的可接收性，我们得出 $p \overset{t \cdot e}{\Downarrow}$，如所需。[Rocq] ◀

▶ **定理 9**。 设 p 具有确定性。那么，$(p \rightrightarrows p') \wedge (p' \rightrightarrows p) \ \Rightarrow p \equiv p'$。 [Rocq] ◀

证明思路。 利用 p的确定性，我们首先证明“半等价性” $\mathrm{Beh}_p(\tau, b) \ \Rightarrow \mathrm{Beh}_{p'}(\tau, b)$。 $p' \rightrightarrows p$ 足以将确定性从 p 传递到 p'，因此我们可以对称地证明另一半等价性，从而完成证明。

假设 $\mathrm{Beh}_p(\tau, b)$。根据 $p \rightrightarrows p'$，对于某个 $(\tau', b') \preccurlyeq (\tau, b)$，有 $\mathrm{Beh}_{p'}(\tau', b')$；根据 $p' \rightrightarrows p$，对于某个 $(\tau'', b'') \preccurlyeq (\tau' b') \preccurlyeq (\tau, b)$，有 $\mathrm{Beh}_p(\tau'', b'')$。然而，由于 p具有确定性，这三个元组必定相等，因为 τ 是 τ'' 的前缀（因为 $\preccurlyeq$ 具有传递性）。[Rocq] ◀

定理 8 是定理 4.5 在行为方面的对应版本；CompCert 使用定理 4.5 将小步前向模拟转换为后向模拟。值得注意的是，两者的前提相同：源语言的可接收性和目标语言的确定性。

2.3 小步语义与大步语义

我们的目标是将小步语义和大步语义纳入上述行为框架。

对于小步语义 $\rightarrow$，我们可以利用其小步行为来定义行为谓词：给定一个初始状态为 σ_p 的程序 p，定义 $p \overset{t}{\Downarrow} :\Leftrightarrow \exists \sigma,\ \sigma_p \xrightarrow{t}{}^* \sigma$ 和 $p \Downarrow^t \sigma :\Leftrightarrow \sigma_p \xrightarrow{t}{}^* \sigma_{fin}$（其中 σ_{fin} 为最终状态）以及 $p \Downarrow^\tau_\infty :\Leftrightarrow \sigma_p \xrightarrow{\tau}_\infty$，其中 $\xrightarrow{\tau}_\infty$ 是第 3 节中定义的发散的余归纳谓词。

▶**引理 10.** 当 $\rightarrow$ 具有确定性时，上述定义会给出一种确定性的行为语义，其行为与 $\rightarrow$ 的小步行为一致。当 $\rightarrow$ 还具有可接收性时，该行为语义也具有可接收性。[Rocq] ◀

[5] 详细证明见我们的 CompCert 扩展 [9]。

之所以需要确定性，是因为否则，仅凭部分执行不足以检测出错：可能存在两次执行 $\sigma_p \xrightarrow{t}{}^{*} \sigma_1 \not\to$ 和 $\sigma_p \xrightarrow{t}{}^{*} \xrightarrow{e}{}^{*} \sigma_2$，在这种情况下，$\mathrm{Beh}_P(t, \mathtt{err})$ 会错误地不成立。在发散情况下，行为与 CompCert 所定义的小步行为一致，这一点并非显而易见——参见定理 12。

Leroy [14] 所描述的大步语义由一个归纳终止判断式和一个余归纳发散判断式定义。要将其纳入行为框架，需要进行两项调整：引入一个部分判断式，并修改发散判断式，使其允许一般轨迹。这两项修改均在第 3 节中介绍。行为大步语义的一项主要优势是，我们无需额外引入复杂的出错判断式，而是可以从其他通常更简单的判断式中推导出它。然而，要证明部分谓词所需的性质对于大步语义而言并非易事。我们可以借助一个等价的小步语义来解决这一问题：如果能够证明相对于行为小步语义的可靠性和完备性，那么大步语义相应的部分性质便可直接得出。

如果 p 是小步语义，p^{BS} 是大步语义，且二者都按行为语义解释，那么 p^{BS} 相对于 p 的可靠性恰好等价于以下三个命题 ($p^{BS} \Downarrow^{t} \sigma \Rightarrow p \Downarrow^{t} \sigma$)、($p^{BS} \Downarrow^{\tau}_{\infty} \Rightarrow p \Downarrow^{\tau}_{\infty}$) 和 ($p^{BS} \Downarrow^{t} \Rightarrow p \Downarrow^{t}$) 的合取。根据引理 7，可得$p^{BS} \rightrightarrows p$。同理，完备性可推出 $p \rightrightarrows p^{BS}$。如果 p 具有确定性，那么定理 9 得出等价关系 $p \equiv p^{BS}$。

► **引理 11.** 确定性与接纳性在等价关系下保持不变。[Rocq] ◄

2.4 流水线集成

我们介绍为集成行为语义而对 CompCert 所做的修改，并指出这一方法也适用于 CompCert 之外的其他小步转换流水线。图 1 顶部展示了一个典型的编译器流水线，从 C 源程序 p_c 开始，到汇编程序 p_{asm}结束。每个变换都会建立一个小步模拟 [11], ，随后将其提升至整个流水线。通过 p_c 与 p_{asm} 之间的小步模拟，可以得到前向保持性 $p_c \rightrightarrows p_{asm}$；再结合语义的其他性质，最终可得反向保持性 $p_c \leftleftarrows p_{asm}$，这也是经验证的编译器的主定理。请注意，$p_c$ 和 p_{asm} 是经典的小步语义，而非行为语义——因此，此处使用记号 $\rightrightarrows$ 和 $\leftleftarrows$ 时，采用的是经典含义，指代小步行为。

图 1 底部展示了我们的流水线。我们使用（确定且可接收的）中间语言 `Cminor`，并将其小步语义和大步语义解释为行为语义。在 `Cminor`处，我们将 CompCert 的流水线拆分为两部分，并转而采用大步语义，以执行各种循环变换 $p^{BS}_{cmin} \to \cdots \to \tilde{p}^{BS}_{cmin}$。之后，我们切回小步语义。小步处理流水线的两部分各自给出一个小步模拟，由此可得反向保持性。根据定理 8，对于 p^{BS}_{cmin} 与$\tilde{p}^{BS}_{cmin}$之间的循环变换，也可得出反向保持性。小步语义与大步语义 `Cminor` 之间的等价性，可由可靠性与完备性以及定理 9 得出。由此得到以下链条

$$p_c \leftleftarrows p_{cmin} \equiv p^{BS}_{cmin} \leftleftarrows \tilde{p}^{BS}_{cmin} \equiv \tilde{p}_{cmin} \leftleftarrows \tilde{p}_{asm}, \text{ and therefore: } p_c \leftleftarrows \tilde{p}_{asm}.$$

这恢复了 CompCert 的主定理。这里再次说明，对于小步语义，$\leftleftarrows$指的是小步行为。我们还为扩展编译器链证明了 CompCert 的所有其他主定理。其中包括关于通过链接进行分离编译的结果 [18, 17]

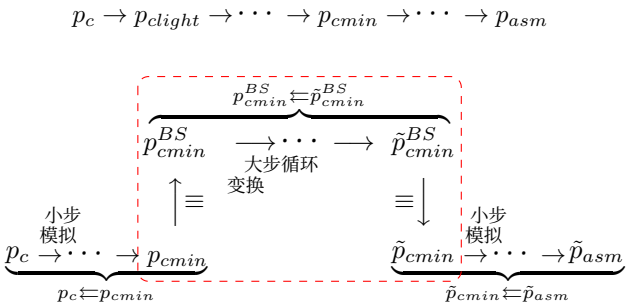

图 1 上方是经典的 CompCert 编译流水线：p_c 被编译为 p_{asm}。下方是新的流水线，它对 `Cminor`执行循环变换。虚线矩形内的部分使用行为框架。

$$\frac{s \xrightarrow{t} s' \quad \mathrm{div}^{\omega} s'}{\mathrm{div}^{t\cdot\omega} s} \qquad \frac{s \xrightarrow{\varepsilon} s' \quad \mathrm{div_silent}^{\varepsilon} s'}{\mathrm{div_silent}^{\varepsilon} s} \qquad \frac{s \xrightarrow{t}{}^{*} s' \quad \mathrm{div_silent}^{\varepsilon} s'}{\mathrm{div_fin}^{t} s} \qquad \frac{s \xrightarrow{t}{}^{*} s' \quad t \neq \varepsilon \quad \mathrm{reacts}^{\omega} s'}{\mathrm{reacts}^{t\cdot\omega} s}$$

图 2 CompCert 中出现的各种小步发散定义。此处及下文中，我们用单线表示归纳判断，用双线表示余归纳判断。

以及关于相对于 CompCert 中实现的另一种 `Cstrategy` 语义的行为的陈述。更多详情可参见我们的开发成果。

3 具有一般轨迹的余归纳发散

给定一个小步语义 $\to$，图 2 左侧展示了经典的发散余归纳定义（div^{ω}），该定义生成一条无限轨迹 ω。这个定义的问题在于，它无法判定 ω 是以余归纳方式完整生成的，还是最终只生成了一个有限前缀 $t < \omega$。例如，一个空的 `while true`-循环可以通过任意无限 ω 发散。可以通过采用多个不同的余归纳定义来解决这一问题，分别定义 silent 发散（$\mathrm{div_silent}^{\varepsilon}$）、有限发散（$\mathrm{div_fin}^{t}$）和无限发散（$\mathrm{reacts}^{\omega}$）；这些定义也展示在图 2 中。$\mathrm{div}^{\omega}$ 无法区分有限发散和无限发散；不过，有如下结论：

$$\mathrm{div}^{\omega} s \Rightarrow \mathrm{reacts}^{\omega} s \vee \exists t < \omega, \mathrm{div_fin}^{t} s. \qquad (*)$$

CompCert 采用了这种方法，并同时使用 $\mathrm{div_fin}^{t}$ 和 reacts^{ω} 来定义小步语义的发散行为。然而，这种方法无法迁移到余归纳大步语义中：首先，所有发散规则都必须重复定义；其次，不清楚如何将 $\mathrm{reacts}^{t\cdot\omega}$ 中带有 $t \neq \varepsilon$ 条件的 $\xrightarrow{t}{}^{*}$ 迁移到大步规则中。

因此，我们将这些定义合并为单一的一个定义，它产生一般[6]（即有限或无限）轨迹 τ，并确保该轨迹确实被完整地产生。

我们的判断式 $\xrightarrow{\tau}_{(n)\infty}$（其定义见图 3）还携带一个自然数 $n \in \mathbb{N}_0$，以确保至多经过 n 步后，τ 的一个非空前缀已被

[6] 我们的一般轨迹不过是有限轨迹与无限轨迹的和类型。这之所以可行，是因为我们可以经典地区分该轨迹是有限的还是无限的。

$$\frac{s \xrightarrow{t} s' \quad s' \xrightarrow[(m)]{\tau}_{\infty} \quad guard\ \tau\ n\ t\ m}{s \xrightarrow[(n)]{t \cdot \tau}_{\infty}}$$

图 3 我们对带有一般轨迹的小步发散的定义。这里，m，$n \in \mathbb{N}_0$ 和 $guard\ \tau\ n\ t\ m := (\tau = \varepsilon) \vee (t = \varepsilon \Rightarrow m < n)$。

生成。当 τ 为有限值时，这一过程会持续到 τ 耗尽；否则则会无限持续。为此，我们定义 $guard\ \tau\ n\ t\ m := (\tau = \varepsilon) \vee (t = \varepsilon \Rightarrow m < n)$。$n$ 的具体值在语义上无关紧要，因此我们有时只写 $\xrightarrow{\tau}_{\infty}$。

▶ **定理 12**。 $s \xrightarrow{\tau}_{\infty} \Rightarrow \begin{cases} div_fin^{\tau}\ s & \text{适用于 } \tau \text{ 有限情形，} \\ reacts^{\tau}\ s & \text{适用于 } \tau \text{ 无限情形。} \end{cases}$

如果 $\rightarrow$ 具有确定性[7]，，则等价关系成立。 [Rocq] ◀

Cousot 和 Cousot [4]最初定义的余归纳大步语义，后经 Leroy [14] 扩展，包含一个用于描述带无限轨迹的发散的余归纳判断式。这会引发与小步语义相同的问题：空循环可以沿任意轨迹发散（$\circlearrowleft$nop） $\Downarrow^{\omega}_{\infty}$。不过，我们可以采用与上文相同的技巧。下图左侧展示了大步语义中的两个余归纳定义，用于刻画循环的发散（完整语义见第4节）：

$$\frac{\langle s,\ \sigma\rangle \Downarrow^{\omega}_{\infty}}{\langle \circlearrowleft s,\ \sigma\rangle \Downarrow^{\omega}_{\infty}} \qquad \frac{\langle s,\ \sigma\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty} \quad guard\ \tau\ n\ \varepsilon\ m}{\langle \circlearrowleft s,\ \sigma\rangle\ {}^{(n)}\!\Downarrow^{\tau}_{\infty}}$$

$$\longrightarrow$$

$$\frac{\langle s,\ \sigma\rangle \Downarrow^{t} \sigma' \quad \langle \circlearrowleft s,\ \sigma'\rangle \Downarrow^{\omega}_{\infty}}{\langle \circlearrowleft s,\ \sigma\rangle \Downarrow^{t \cdot \omega}_{\infty}} \qquad \frac{\langle s,\ \sigma\rangle \Downarrow^{t} \sigma' \quad \langle \circlearrowleft s,\ \sigma'\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty} \quad guard\ \tau\ n\ t\ m}{\langle \circlearrowleft s,\ \sigma\rangle\ {}^{(n)}\!\Downarrow^{t \cdot \tau}_{\infty}}$$

循环既可以在循环体发散时发散（$\Downarrow^{\omega}_{\infty}$），也可以在循环体终止（$\Downarrow^{t}$）后继续发散。按照余归纳的方式解释，这也包括终止的循环体无限执行的情况。我们通过添加刚引入的 $guard$条件，更新右侧所示的每个判断式。如果规则应用本身无法生成轨迹，我们就在守卫中使用 ε；否则，我们使用生成的轨迹 t。这保证了在应用发散规则至多 n 次后，已经生成了一般轨迹 τ的非空前缀；但当 τ 为空时例外，因此确保完整生成现在的一般轨迹 τ。与小步情形一样，$n \in \mathbb{N}_0$ 的实际值无关紧要，因此我们会交替使用 ${}^{(n)}\!\Downarrow^{\tau}_{\infty}$ 和 $\Downarrow^{\tau}_{\infty}$。

虽然我们不需要确立新旧大步判断式之间的任何关系，但通过余归纳很容易证明以下结论：

$$\forall \tau \le \omega,\ \langle s,\ \sigma\rangle\ {}^{(n)}\!\Downarrow^{\tau}_{\infty} \Rightarrow \langle s,\ \sigma\rangle \Downarrow^{\omega}_{\infty}.$$

对于确定性语义，例如 `Cminor`，证明另一个方向（给定 ω，$\exists \tau \le \omega$）也是可行的。不过，这个证明并非直接得出，而是通过新的和旧的

[7] 给定带有 $t \neq \varepsilon$ 的 $\xrightarrow{t}{}^{*}$，需要确定性来唯一地确定第一个事件发生时的步数，这是 $\xrightarrow{\tau}_{\infty}$ 的余归纳定义所必需的。

小步语义（它们分别等价于相应的大步语义），并应用上述等式（$*$）和定理 12 来在两者之间进行转换。

4 Cminor

`Cminor`是 CompCert 的一种早期中间表示，由表达式、语句和函数组成 [1]。函数并非由控制流图定义，而是由单个归纳定义的语句定义。下面给出修改后的 `Cminor` 的一个节选（为简洁起见，省略了内部函数调用、内存及其他内容），本文余下部分将使用它。它包含以下语句：

$$
\begin{array}{lll}
stmt ::= & \mathsf{nop} & \\
& \mid reg \leftarrow expr & \text{寄存器操作} \\
& \mid expr\ ?\ stmt : stmt & \text{条件语句} \\
& \mid stmt \cdot\ stmt & \text{顺序执行} \\
& \mid \circlearrowleft stmt & \text{loop} \\
& \mid [\![stmt]\!] & \text{包裹在块中} \\
& \mid \mathsf{Exit}\, nat & \text{退出 } n+1 \text{ 外围块} \\
& \mid \text{不安全 } function & \text{调用外部函数}
\end{array}
$$

不安全语句是唯一会产生事件的语句。循环必须放在块中，才能通过退出语句跳出循环。例如，以下语句描述了一个执行 $stmt$ 5 次的循环：

$$
\begin{aligned}
i \leftarrow 0 \cdot [\![\circlearrowleft \big(\ & (i <^{?} 5) \\
& ?\ stmt \cdot (i \leftarrow i+1) \\
& :\ \mathsf{exit}\, 0\ \big)]\!]
\end{aligned}
$$

图 4 展示了小步语义和大步语义。小步状态的形式为$\langle stmt,\ cont,\ \sigma \rangle$，其中，$cont$ 是执行完 $stmt$ 后接下来要执行的续延，而 σ 包含全局环境和寄存器状态。续延可以是 stop（执行结束）、$stmt \cdot cont$（先执行 $stmt$，然后执行 $cont$），或 $[\![cont]\!]$（将$cont$包在一个块中）。[8]

大步语义由两个判断式组成：归纳判断式$\langle stmt,\ \sigma \rangle \Downarrow^{t} out$和余归纳判断式$\langle stmt,\ \sigma \rangle {}^{(n)}\!\Downarrow^{\tau}_{\infty}$。语句执行时会产生一个结果，它可以是 `OK` σ'（正常执行并到达最终状态 σ'）、`Exit` n σ'（退出 $n \in \mathbb{N}_0$所在的外围块，其中 $n+1$）或 `Partial`。

我们通过两种方式修改了原始的大步语义：首先，我们在每条发散规则中加入了 $guard$语句，以允许出现具有一般轨迹的发散行为，如第 3 节所述。其次，我们加入了两条产生 `Partial` 结果的规则，从而将部分求值和完全求值结合到同一个判断式中。

请注意，要得到 `Partial` 结果，必须先生成完整轨迹，之后才能使用规则 $s \Downarrow^{\varepsilon}$ `Partial` 将求值“截断”。根据定义可知，任何完全求值都会导出一个具有相同轨迹的部分求值。在确立了相对于小步语义的可靠性和完备性之后，行为语义所需的其他所有性质也随之成立，如第 2.3 节所述。

[8] 在 CompCert `Cminor` 中，栈轨迹也完全编码在续延中。

10 结合小步与大步语义验证循环优化

$$\frac{\text{是_真（求值 } \sigma\ e)}{\langle e\ ?\ \ s_1 :\ \ s_2, k,\ \sigma\rangle \xrightarrow{\varepsilon} \langle s_1,\ k,\ \sigma\rangle} \qquad \frac{\text{是_假（求值 } \sigma\ e)}{\langle e\ ?\ \ s_1 :\ \ s_2, k,\ \sigma\rangle \xrightarrow{\varepsilon} \langle s_2,\ k,\ \sigma\rangle}$$

$$\frac{\text{求值 } \sigma\ e = v}{\langle r \leftarrow e,\ k,\ \sigma\rangle \xrightarrow{\varepsilon} \langle \mathsf{nop},\ \ k,\ \sigma[r := v]\rangle} \qquad \frac{\text{外部_调用 } f\ \sigma = \sigma'}{\langle \text{不安全 } f, k,\ \sigma\rangle \xrightarrow{\mathtt{CALL}\ f\ \sigma} \langle \mathsf{nop},\ \ k,\ \sigma'\rangle}$$

$$\langle \circlearrowleft s,\ k,\ \sigma\rangle \xrightarrow{\varepsilon} \langle s,\ (\circlearrowleft s) \cdot k, \sigma\rangle \qquad \langle s_1 \cdot s_2,\ k,\ \sigma\rangle \xrightarrow{\varepsilon} \langle s_1,\ s_2 \cdot k, \sigma\rangle$$

$$\langle \mathsf{nop}\ \ s \cdot k, \sigma\rangle \xrightarrow{\varepsilon} \langle s,\ k,\ \sigma\rangle \qquad \langle \mathsf{nop},\ \ [\![k]\!],\ \sigma\rangle \xrightarrow{\varepsilon} \langle \mathsf{nop},\ \ k,\ \sigma\rangle$$

$$\langle [\![s]\!],\ k,\ \sigma\rangle \xrightarrow{\varepsilon} \langle s,\ [\![k]\!],\ \sigma\rangle \qquad \langle \text{退出 } n,\ \ s \cdot k, \sigma\rangle \xrightarrow{\varepsilon} \langle \text{退出 } n, k,\ \sigma\rangle$$

$$\langle \text{退出 } 0,\ \ [\![k]\!],\ \sigma\rangle \xrightarrow{\varepsilon} \langle \mathsf{nop},\ \ k,\ \sigma\rangle \qquad \langle \text{退出 } (n+1),\ [\![k]\!],\ \sigma\rangle \xrightarrow{\varepsilon} \langle \text{退出 } n, k,\ \sigma\rangle$$

小步判断 $\xrightarrow{t}$

$$\forall s:\ \ s \Downarrow^{\varepsilon} \mathtt{Partial} \qquad \langle \mathsf{nop}\ \ \sigma\rangle \Downarrow^{\varepsilon} \mathtt{OK}\,\sigma \qquad \langle \text{退出 } n,\ \ \sigma\rangle \Downarrow^{\varepsilon} \mathtt{Exit}\ n\ \sigma$$

$$\frac{\text{为_真（求值 } \sigma\ e)\ \ \langle s_1,\ \sigma\rangle \Downarrow^{t} o}{\langle e\ ?\ \ s_1 :\ \ s_2, \sigma\rangle \Downarrow^{t} o} \qquad \frac{\text{是_假（求值 } \sigma\ e)\ \ \langle s_2,\ \sigma\rangle \Downarrow^{t} o}{\langle e\ ?\ \ s_1 :\ \ s_2, \sigma\rangle \Downarrow^{t} o}$$

$$\frac{\text{求值 } \sigma\ e = v}{\langle r \leftarrow e,\ \sigma\rangle \Downarrow^{\varepsilon} \mathtt{OK}\,\sigma[r := v]} \qquad \frac{\text{外部_调用 } f\ \sigma = \sigma'}{\langle \text{不安全 } f,\ \ \sigma\rangle \Downarrow^{\mathtt{CALL}\ f\ \sigma} \mathtt{OK}\,\sigma' \vee \mathtt{Partial}^{(*)}}$$

$$[\circlearrowleft_{exit}]\frac{\langle s,\ \sigma\rangle \Downarrow^{t} o \qquad o \neq \mathtt{OK}\ _}{\langle \circlearrowleft s,\ \sigma\rangle \Downarrow^{t} o} \qquad [\circlearrowleft_{cont}]\frac{\langle s,\ \sigma_1\rangle \Downarrow^{t_1} \mathtt{OK}\,\sigma_2 \qquad \langle \circlearrowleft s,\ \sigma_2\rangle \Downarrow^{t_2} o}{\langle \circlearrowleft s,\ \sigma_1\rangle \Downarrow^{t_1 \cdot t_2} o}$$

$$[\cdot_{exit}]\frac{\langle s_1,\ \sigma\rangle \Downarrow^{t} o \qquad o \neq \mathtt{OK}\ _}{\langle s_1 \cdot s_2, \sigma\rangle \Downarrow^{t} o} \qquad [\cdot_{cont}]\frac{\langle s_1,\ \sigma_1\rangle \Downarrow^{t_1} \mathtt{OK}\,\sigma_2 \qquad \langle s_2,\ \sigma_2\rangle \Downarrow^{t_2} o}{\langle s_1 \cdot s_2, \sigma_1\rangle \Downarrow^{t_1 \cdot t_2} o}$$

$$[[\![]\!]^1_{exit}]\frac{\langle s,\ \sigma\rangle \Downarrow^{t} \mathtt{OK}\,\sigma' \vee \mathtt{Exit}\ 0\ \sigma'}{\langle [\![s]\!],\ \sigma\rangle \Downarrow^{t} \mathtt{OK}\,\sigma'} \qquad [[\![]\!]^2_{exit}]\frac{\langle s,\ \sigma\rangle \Downarrow^{t} \mathtt{Exit}\ (n+1)\ \ \sigma'}{\langle [\![s]\!],\ \sigma\rangle \Downarrow^{t} \mathtt{Exit}\ n\ \sigma'}$$

归纳式部分与完整执行判断 $\Downarrow^{t}$ (*) 这是两条独立的规则

.

$$\frac{\text{is_true (求值 } \sigma\ e)\ \ \langle s_1,\ \sigma\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty}}{\langle e\ ?\ \ s_1 : s_2, \sigma\rangle\ {}^{(n)}\!\Downarrow^{\tau}_{\infty}} \qquad \frac{\text{is_false (求值 } \sigma\ e)\ \ \langle s_2,\ \sigma\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty}}{\langle e\ ?\ \ s_1 :\ \ s_2, \sigma\rangle\ {}^{(n)}\!\Downarrow^{\tau}_{\infty}}$$

$$[[\![]\!]_{div}]\frac{\langle s,\ \sigma\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty}}{\langle [\![s]\!],\ \sigma\rangle\ {}^{(n)}\!\Downarrow^{\tau}_{\infty}} \qquad [\cdot_{div}]\frac{\langle s_1,\ \sigma\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty}}{\langle s_1 \cdot\ s_2,\ \sigma\rangle\ {}^{(n)}\!\Downarrow^{\tau}_{\infty}} \qquad [\circlearrowleft_{div}]\frac{\langle s,\ \sigma\rangle \Downarrow^{\tau}_{\infty} m}{\langle \circlearrowleft s,\ \sigma\rangle\ \Downarrow^{\tau}_{\infty} n}$$

$$[\cdot'_{cont}]\frac{\langle s_1,\ \sigma_1\rangle \Downarrow^{t} \mathtt{OK}\,\sigma_2 \qquad \langle s_2,\ \sigma_2\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty}}{\langle s_1 \cdot s_2, \sigma_1\rangle\ {}^{(n)}\!\Downarrow^{t \cdot \tau}_{\infty}} \qquad [\circlearrowleft'_{cont}]\frac{\langle s,\ \sigma_1\rangle \Downarrow^{t} \mathtt{OK}\,\sigma_2 \qquad \langle \circlearrowleft s,\ \sigma_2\rangle\ {}^{(m)}\!\Downarrow^{\tau}_{\infty}}{\langle \circlearrowleft s,\ \sigma_1\rangle\ {}^{(n)}\!\Downarrow^{t \cdot \tau}_{\infty}}$$

余归纳发散判断 ${}^{(n)}\!\Downarrow^{\tau}_{\infty}$。每条规则中均省略了 *guard* 前提。

图 4 `Cminor` 的小步语义与大步语义（节选）

4.1 可靠性和完备性

可靠性的大步 Cminor 意味着，每个终止、部分或发散的大步判断式都会产生相应的小步执行。这可以通过归纳或反归纳证明，而 CompCert 已经针对经典的 Cminor完成了这项工作。由于它不足以区分有限轨迹和无限轨迹发散，可靠性陈述相应地较弱，只给出了一个发$^\omega$ 结果。借助我们的一般大步发散，我们能够轻松地将其加强为 $\xrightarrow{\tau}_\infty$，从而根据定理 12 得到正确的小步行为。此外，我们几乎不费吹灰之力就证明了部分情形，只需将其纳入终止情形即可。

完备性对于终止而言，意味着给定任意终止的小步执行$\langle s,\ k,\ \sigma\rangle \xrightarrow{t}{}^*\langle \mathsf{nop},\ \mathsf{stop}\ \sigma_{fin}\rangle$，语句 s 必须成功求值，因此在原始状态和最终状态之间必定存在一个“终止”状态，它对应于对 s的大步求值：

$$\langle s,\ k,\ \sigma\rangle \xrightarrow{t_1}{}^*\langle \mathsf{nop},\ k,\ \sigma'\rangle \xrightarrow{t_2}{}^*\langle \mathsf{nop},\ \mathsf{stop},\ \sigma_{fin}\rangle \quad \text{with } t = t_1 \cdot t_2 \text{ and } \langle s,\ \sigma\rangle \Downarrow^{t_1} \mathtt{OK}\,\sigma'.$$

这可以通过对（有限）执行进行反向归纳来证明。以极为类似的方式，我们可以证明，任何小步执行$\langle s,\ k,\ \sigma\rangle \xrightarrow{t}{}^*\langle s',\ k',\ \sigma'\rangle$ 要么对应于对 s **(**$s \Downarrow^t$ `Partial`**)** 正在进行的部分求值，要么是 s 的某个终止状态存在于该执行之中，并伴随对 s的完整大步求值，正如上文所述。将这两个语句应用于初始状态，便得到我们所期望的 $\Downarrow$ 与 的完备性。

如果某个**发散的**小步执行访问状态$\langle s,\ k,\ \sigma\rangle$，则要么是 s 本身导致了发散，要么是 s 终止而 k 内部的某处发散。因此，我们希望证明：

$$\text{Given } \langle s,\ k,\ \sigma\rangle \xrightarrow[(n)]{\tau}_\infty, \text{ either } \left(\langle s,\ \sigma\rangle \overset{(n)}{\Downarrow}{}^\tau_\infty\right) \text{ or } \left(\exists (t \le \tau), (out \neq \mathtt{Partial}) : \langle s,\ \sigma\rangle \Downarrow^t out\right)$$

该语句尚不能以共归纳方式证明，因为其结论中含有 $\vee$ 和 $\exists$ 。不过，将结论的第二种情形取否定并作为前提移到左侧，便可得到所需形式。将其应用于初始状态，并结合可靠性与确定性，我们得以得出 $\Downarrow_\infty$的完备性。由于 Cminor 是确定性且可接收的，定理 9 给出行为等价，而引理 11 则给出大步 Cminor的确定性与可接收性。

5 循环变换

Cminor 由于其归纳性质，尤其是归纳定义的循环语句，它非常适合用大步语义进行推理：变换很容易定义，其大步证明通常也十分自然。这里我们展示并讨论一些已实现的循环变换。[9]

5.1 循环外提

循环外提是一种变换：当分支与循环体无关时，将分支提升到循环之外。我们只关注一种简单情况：循环体一开始就是这个无关分支，并被该分支分成两部分。我们将这样的循环称为“相关循环”。

[9] 不过，大步语义本身并不支持 goto 语句。因此，如果存在 goto 语句，我们的开发工具允许用户跳过大步阶段。

变换 f： $stmt \to stmt$ 按语句结构归纳定义：

$$
\begin{aligned}
f(\circlearrowright(c\ ?\ s_1 : s_2)) &= c\ ?\ \circlearrowright(fs_1) : \circlearrowright(fs_2) && \text{if } indep\ c\ s_1 \wedge indep\ c\ s_2\\
f(\circlearrowright s) &= \circlearrowright(fs) && \text{for other loops}\\
f(c\ ?\ s_1 : s_2) &= c\ ?\ fs_1 : fs_2\\
f(s_1 \cdot s_2) &= fs_1 \cdot fs_2\\
f([\![s]\!]) &= [\![fs]\!]\\
fs &= s && \text{for all other } s.
\end{aligned}
$$

因此， f 会转换所有相关的循环，包括嵌套循环，无论它们在语句结构中的嵌套深度有多深。谓词 $indep\ \ e\ s$ 表示该语句s 不会写入表达式 e所使用的任何寄存器。

▶ **引理 13** (indep_spec)。 $indep\ \ c\ s \wedge \langle s,\ \sigma\rangle \Downarrow^t \mathtt{OK}\sigma' \Rightarrow$ 求值 $\ c\ \sigma =$ 求值 $\ c\ \sigma'$。

这是联系 $indep$ 与 $eval$ 的主要引理，并通过对 $\Downarrow^t$进行归纳证明。

要使用大步语义证明该变换的正确性——也就是说， f 在程序及其转换版本之间实现前向保持$p \rightrightarrows fp$ ——只需证明这两个语句 $s,\ \sigma\ \Downarrow^t out \Rightarrow fs,\ \sigma\ \Downarrow^t out$和 $s,\ \sigma\ \Downarrow^\tau_\infty \Rightarrow fs,\ \sigma\ \Downarrow^\tau_\infty$。为证明 $s,\ \sigma\ \Downarrow^t out \Rightarrow fs,\ \sigma\ \Downarrow^t out$ ，我们对 $\Downarrow^t$ 进行结构归纳，这直接给出所有非特殊循环的 s 的结果。

现在考虑一个特殊循环 $\langle\circlearrowright(c\ ?\ s_1 :\ s_2)\ \ \sigma\rangle\ \Downarrow^t o$。我们需要证明 $\langle c\ ?\ \ \circlearrowright(fs_1) :\circlearrowright(fs_2)\ \ \sigma\ \rangle\ \Downarrow^t o$。该假设可能出现在两种情况下：循环在第一次迭代期间通过规则 $[\circlearrowright_{exit}]$退出，或者在后续迭代中通过规则 $[\circlearrowright_{cont}]$退出。第一种情况如下图所示：

$$
\uparrow\ \dfrac{\dfrac{\dfrac{\overbrace{\langle f(c\ ?\ s_1 : s_2),\ \sigma\rangle}^{c\ ?\ fs_1 : fs_2} \Downarrow^t o}{\langle c\ ?\ s_1 : s_2,\ \sigma\rangle \Downarrow^t o}\ \text{by ind} \qquad o \neq \mathtt{OK}_}{\langle\circlearrowright(c\ ?\ s_1 : s_2),\ \sigma\rangle \Downarrow^t o}\ [\circlearrowright_{exit}]}{\text{is_true (eval } c\ \sigma) \qquad \langle fs_1,\ \sigma\rangle \Downarrow^t o}\ \text{wlog}
\qquad\qquad
\downarrow\ \dfrac{\text{is_true (eval } c\ \sigma) \qquad \dfrac{\langle fs_1,\ \sigma\rangle \Downarrow^t o \neq \mathtt{OK}_}{\langle\circlearrowright(fs_1),\ \sigma\rangle \Downarrow^t o}}{\langle c\ ?\ \circlearrowright(fs_1) : \circlearrowright(fs_2),\ \sigma\rangle \Downarrow^t o}
$$

假设显示在左下方。从这里开始，我们通过展开大步判断向上推导。在虚线处，使用归纳假设将 f 移入语句，同时不改变结果。对于所得 if 语句的执行，我们不失一般性地假设条件 c 求值为 true；另一种情况类似。利用由此得到的事实，即可在右侧自上而下地证明所需语句。

我们只需对判断式作简单展开并使用归纳，就证明了这种情况。第二种情况稍微有趣一些：循环完整执行一次迭代后，再递归求值。如下图所示：

$$
\uparrow\ \dfrac{\dfrac{\dfrac{\langle c\ ?\ fs_1 : fs_2,\ \sigma\rangle \Downarrow^{t_1} \mathtt{OK}\,\sigma'}{\langle c\ ?\ s_1 : s_2,\ \sigma\rangle \Downarrow^{t_1} \mathtt{OK}\,\sigma'}\ \text{by ind} \qquad \dfrac{\overbrace{\text{is_true (eval } c\ \sigma')}^{=\ \text{eval } c\ \sigma} \qquad \langle\circlearrowright(fs_1),\ \sigma'\rangle \Downarrow^{t_2} o}{\langle c\ ?\ \circlearrowright(fs_1) : \circlearrowright(fs_2),\ \sigma'\rangle \Downarrow^{t_2} o}\ \text{by ind}\ (*)}{\langle\circlearrowright(c\ ?\ s_1 : s_2),\ \sigma'\rangle \Downarrow^{t_2} o}}{\langle\circlearrowright(c\ ?\ s_1 : s_2),\ \sigma\rangle \Downarrow^{t_1 \cdot t_2} o}\ [\circlearrowright_{cont}]
$$

通过两次归纳，我们得到了关于转换后的前提执行情况的两个结论。引理 13 得出求值 $c\ \sigma' =$eval $\ c\ \sigma$。因此，在 $(*)$ 处只可能出现 is_true 的情况

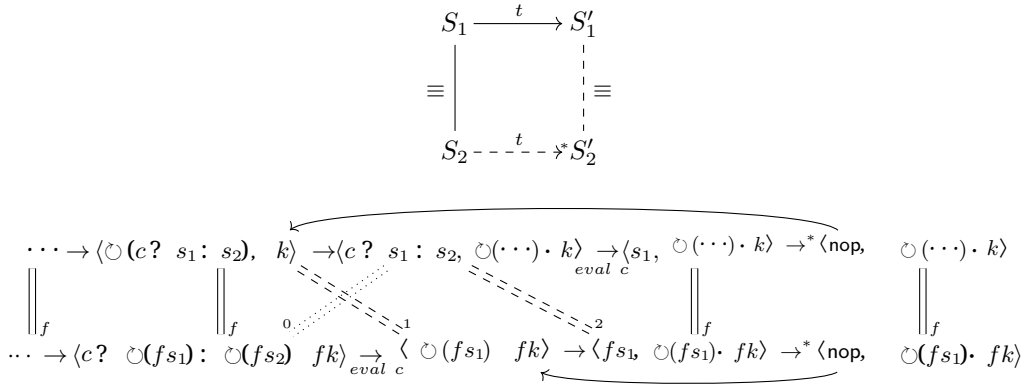

图5 上方是一个通用的小步模拟图。下方是循环外提的小步模拟图。原始程序的执行直接显示在变换后的执行上方；竖直双线表示哪些状态需要彼此匹配。为简洁起见，省略了环境和轨迹。

（左上方已不失一般性地假定了这一点），这为我们提供了有关 $\circlearrowright(fs_1)$ 的另一次执行的重要信息，而不只是可能的 $\circlearrowright(fs_2)$。然后，根据关于求值 $c\,\sigma$ 以及 fs_1 和 $\circlearrowright(fs_1)$ 的执行所获得的事实，即可立即得出所需结论。

这证明了终止情形下的保持性，实际上也顺带确立了部分情形。利用对 $\Downarrow_\infty^\tau$ 的余归纳来证明发散情形，思路十分相似；对于大多数语句，这会直接得出结果，而对于特殊循环，则需考察 $[\circlearrowright_{div}]$ and$[\circlearrowright'_{cont}]$ 这两种情形。这里不作展开，只指出：要在 Rocq 中证明这一点，需要额外选取一个恰当的余归纳假设，才能通过 Rocq 的语法守卫检查器；在某些情况下，这可能相当棘手。

小步

现在，我们概述以小步方式证明 f 的正确性。因此，必须建立一个小步模拟，并在源状态与目标状态之间建立恰当的匹配关系 [11]。图5 上方展示了小步模拟的含义：给定两个匹配状态 $S_1 \equiv S_2$，以及源程序中的一步执行 $S_1 \xrightarrow{t} S_2$，必须找到一个状态 S_2'，使得 $S_2 \xrightarrow{t}{}^* S_2'$ 且 $S_1' \equiv S_2'$。步骤之间不必严格一一对应：对于源程序中的一步执行，目标程序可以执行零步或多步。

首先，必须定义原始程序 p 与变换后的程序 fp中状态之间的匹配谓词 $=_f$。在特殊循环之外，这只是 $s, k, \sigma =_f fs, fk, \sigma$，其中我们将 f 归纳地提升到续延上。遇到特殊循环时，情况就变得有趣了。这个过程在图5底部有所展示：进入循环之前，执行处于图左侧所示的位置，即 $\circlearrowright(c\,?\;s_1 : s_2)\;\; k, \sigma =_f c\,?\;\circlearrowright(fs_1) : \circlearrowright(fs_2)\;\; fk, \sigma$。在 p中，下一步会先展开循环，之后才对 c求值。相比之下，在 fp中会立即对 c 求值。因此，fp 必须等到 p 执行两步之后；需要谓词 $=_0$ 来允许匹配中间状态。一旦在 p中对 c 求值，就可以在 fp中一次执行两步，随后恢复标准的 $=_f$匹配，并在循环剩余部分中始终保持这种匹配。

I如果循环正常执行（即未被 Exit 终止且没有发散），

在 p 和 fp 中的执行都会到达一个空操作语句，如图右侧所示。从这里开始， p 和 fp 都会返回各自循环的开头。然而，在 $=_f$下，这些状态并不匹配，因此我们还要求 $=_1$ 和 $=_2$ 在返回循环体之前保持匹配；这些循环体又会通过 $=_f$匹配。

为了正确考虑特殊循环，现在必须将匹配定义也扩展到续延上，从而携带有关 c 求值的信息，以及该求值与当前语句和续延相关部分的独立性，以证明在 p 中，求值 $c\,\sigma$ 始终会得出 true（或 false）。执行特殊循环期间，续延之间的匹配可能如下所示（其中 a_i 是 s_1 的子语句）：

$$a_n \cdot \ldots \cdot a_1 \cdot \circlearrowleft(c\ ?\ s_1 : s_2) \cdot k \cong_\sigma fa_n \cdot \ldots \cdot fa_1 \cdot \circlearrowleft(fs_1) \cdot fk,$$
$$given\ \text{is_true}\ (\text{eval}\ c\ \sigma) \wedge indep\ c\ s_1 \wedge \forall i, indep\ c\ a_i.$$

要正确考虑嵌套 特殊循环，会变得棘手得多，因为多个不同条件会分别独立于续延的不同部分。

有了完整的匹配定义，就可以证明 p 中的每一步都对应于 fp 中的一步或两步，或者 p 中的两步对应于 fp 中的一步，并且匹配以及 $indep$ 和 $eval$ 不变式都得以保持。由此可得小步模拟，进而确立前向保持性 $p \rightrightarrows fp$。

这种匹配定义之所以复杂，部分原因在于 Cminor 是通过归纳定义的，而不是通过控制流图定义的。尽管如此，匹配谓词通常必须同时满足多项功能：除了匹配状态之外，它还必须携带 p 与 fp之间成立的所有不变式。因此，匹配谓词和实际的模拟证明往往都会变得臃肿。请注意，小步证明并未使用引理 13（对 $indep$的紧凑规约）。相反，在模拟证明的每一部分中——也就是对于每一种不同的步匹配组合——都必须证明，对于该特定情形，独立性和 $eval$ 的结果均得以保持。这样实际上就把引理 13 拆散并散布到了整个模拟证明中，削弱了关注点分离，也减少了证明（即代码）复用的机会。

5.2 循环展开

```
int x=1;
for (int i=1; i<11; i++)
    x = x*i;
return x;
```

```
int x=1; int i=1;
x = x*i;
i++;
...
x = x*i;
i++;
return x;
```

```
return 3628800;
```

图 6 左：计算 10! 的循环。中：循环展开后的代码。右：循环展开并经过 CompCert 的常量传播后得到的代码。

图 6 展示了我们经过验证的循环展开过程，以及它如何与常量传播协同工作，静态计算出 10! 的值。一般而言，考虑如下形式的循环：*for (i=0; i<**m**; i++) do stmt* 其中$\mathbf{m}$ 是常量。在 Cminor 中，循环体可能如下所示： $body_m := \big((i <^? \mathbf{m}\ ?\ \text{Exit 0} : \text{nop}) \cdot stmt \cdot (i \leftarrow i{+}1)\big)$. 该变换处理如下形式的语句

$(i \leftarrow 0) \cdot [\![\circlearrowleft body_m]\!]$，并将其展开为 $(i \leftarrow 0) \cdot \mathrm{repeat}_m(stmt \cdot (i \leftarrow i+1))$，其中 $\mathrm{repeat}_0\, s := \mathrm{nop}$ 且 $\mathrm{repeat}^{i+1}\, s := s \cdot \mathrm{repeat}_i\, s$。与前文一样，我们按照语句结构归纳定义该变换$f$；找到匹配的语句并满足若干条件时，便执行循环展开。最重要的条件包括：条件 $i <^? \mathbf{m}$与 $stmt$ 无关——这里沿用之前的 $indep$ 谓词——并且 $stmt$ 本身不会退出循环。为避免代码爆炸，展开循环的循环体本身不会递归展开。

为了证明大步风格下的保持性，我们首先引入一个概念 $\overset{\circlearrowleft}{n}\Downarrow^t$。与 `Exit` 或 `Partial` 结果搭配使用时，它表示循环已退出，或求值在第 n 次迭代期间被截断：

$$\frac{\langle s,\ \sigma\rangle \Downarrow^t o}{\langle s,\ \sigma\rangle \overset{\circlearrowleft}{0}\Downarrow^t o} \qquad \frac{\langle s,\ \sigma\rangle \Downarrow^{t_1} \mathtt{OK}\,\sigma' \qquad \langle s,\ \sigma'\rangle \overset{\circlearrowleft}{i}\Downarrow^{t_2} o}{\langle s,\ \sigma\rangle \overset{\circlearrowleft}{i+1}\Downarrow^{t_1 \cdot t_2} o}$$

类似地，我们引入一个表示发散的概念 $\overset{\circlearrowleft}{n}\Downarrow^\tau_\infty$，意为循环体在第 n 次迭代期间发生了发散。随后，保持性的证明分为以下四个主要步骤：

1. 针对 $body_m$，展示关于单次执行的若干基本结论（关于 $\sigma[i]$ 的值）。**2.** 精确统计已执行的循环迭代次数：给定 $\sigma[i] = 0$,

(a) $\langle \circlearrowleft body_m,\ \sigma\rangle \Downarrow^t \mathtt{Exit}\ 0\ \sigma \Rightarrow \langle body_m,\ \sigma\rangle \overset{\circlearrowleft}{m}\Downarrow^t \mathtt{Exit}\ 0\ \sigma$ ***(b)*** $\langle \circlearrowleft body_m,\ \sigma\rangle \Downarrow^t \mathtt{Partial} \Rightarrow \exists l \le m, \langle body_m,\ \sigma\rangle \overset{\circlearrowleft}{l}\Downarrow^t \mathtt{Partial}$ ***(c)*** $\langle \circlearrowleft body_m,\ \sigma\rangle \Downarrow^\tau_\infty \Rightarrow \exists l \le m, \langle body_m,\ \sigma\rangle \overset{\circlearrowleft}{l}\Downarrow^\tau_\infty$

3. 按其执行次数展开循环：

(a) $\langle body_m,\ \sigma\rangle \overset{\circlearrowleft}{m}\Downarrow^t \mathtt{Exit}\ 0\ \sigma \Rightarrow \langle \mathrm{repeat}_m\,(stmt \cdot (i \leftarrow i+1)),\ \sigma\rangle \Downarrow^t \mathtt{OK}\,\sigma$

(b) $\forall l \le m, \langle body_m,\ \sigma\rangle \overset{\circlearrowleft}{l}\Downarrow^t \mathtt{Partial} \Rightarrow \langle \mathrm{repeat}_l\,(stmt \cdot (i \leftarrow i+1)),\ \sigma\rangle \Downarrow^t \mathtt{Partial}$

(c) $\forall l \le m, \langle body_m,\ \sigma\rangle \overset{\circlearrowleft}{l}\Downarrow^\tau_\infty \Rightarrow \langle \mathrm{repeat}_l\,(stmt \cdot (i \leftarrow i+1)),\ \sigma\rangle \Downarrow^\tau_\infty$

4. 如有必要，进一步展开（共m 次）：

(a) $\forall s,\ i \le j,\ \langle \text{重复}_i\, s,\ \ \sigma\rangle \Downarrow^t \mathtt{Partial} \Rightarrow \langle \text{重复}^j\, s,\ \ \sigma\rangle \Downarrow^t \mathtt{Partial}$

(b) $\forall s,\ i \le j,\ \langle \text{重复}_i\, s,\ \ \sigma\rangle \Downarrow^\tau_\infty \Rightarrow \langle \text{重复}^j\, s,\ \ \sigma\rangle \Downarrow^\tau_\infty$

如果循环 $\circlearrowleft body_m$ 终止，则它恰好对应 m 次循环迭代（给定 $\sigma[i] = 0$）。考虑发散或部分执行时，情况并非如此：在这些情况下，在循环体分别发散或终止被截断之前，迭代次数至多为m。这要求我们分别证明终止、部分执行和发散情形下的若干语句。将它们改写为稍微一般化的形式后，可以结合步骤 1 中建立的引理，对 $\sigma[i]$ 的初始值进行反向归纳，从而归纳证明步骤 2 和 3 中的语句。请注意，循环不可能以 `OK` 结果终止，因此只有 `Exit` 相关。步骤 1 中，使用引理 13 作为关键依据来推理 $\sigma[i]$ 的值。

综合起来，我们得到所需的保持性：$(i \leftarrow 0) \cdot [\![\circlearrowleft body_m]\!]$ 的任意行为都会导出展开循环 $(i \leftarrow 0) \cdot \mathrm{repeat}_m(stmt \cdot (i \leftarrow i+1))$ 的相同行为。对于非循环语句，由于归纳定义，保持性很直接——我们只需稍微注意一下（共）归纳假设，因为我们选择不递归地展开循环。

5.3 While True

大步方法的一个有趣小例子，是一种消除永不退出的静默循环循环体的变换。当一条语句从不发出任何事件时，它就是 silent 的；如果不包含退出语句，它就无法退出。因此，循环 $\circlearrowleft s$ 被转换为 $\circlearrowleft$ nop,

给定 $noexit\ s \wedge silent\ s$。尽管有人可能认为这种变换毫无意义，但它仍然是正确的：它细化了代码的行为。

▶ **引理 14**。 $noexit\ s \wedge \langle \circlearrowleft s,\ \sigma \rangle \Downarrow^{t} o \Rightarrow o = \mathtt{Partial}$

▶ **引理 15**。 $silent\ s \Rightarrow \left([\langle s,\ \sigma \rangle \Downarrow^{t} o \Rightarrow t = \varepsilon] \wedge [\langle s,\ \sigma \rangle \Downarrow_{\infty}^{\tau} \Rightarrow \tau = \varepsilon]\right)$

借助上述刻画 $noexit$ 和 $silent$ 的引理，大步保持性的证明几乎变得轻而易举。尤其是，由于引理 14，终止情形无需特别考虑。对于 Partial 和发散情形，我们利用轨迹为空这一事实（引理 15）。简单事实 $s \Downarrow^{\varepsilon} \mathtt{Partial}$ 和 $\circlearrowleft \mathrm{nop} \Downarrow_{\infty}^{\varepsilon}$ 随即完成证明。

6 讨论与相关工作

我们认为，两种语义形式各有优势，应在最适合的场景中使用。虽然小步语义适用于许多变换，但对于循环优化这类结构变换，大步语义更为自然。像 `Cminor` 这样归纳定义的语言非常适合这类变换。其优势之一是，正确性论证的大部分内容都可以通过结构归纳直接得出。此外，大步推理避免了逐一匹配归约步的需要。因此，无需构造复杂的匹配关系，也不必显式地在线程中传递关于中间状态的不变式。相反，证明通常可以分解为更小的逻辑部分，从而提高模块性和可复用性。例如，引理 13（indep_spec）是循环外提和循环展开的重要组成部分。

然而，大步证明需要重复付出努力，因为归纳陈述和余归纳陈述无法合并为一个陈述。此外，虽然有时可以将部分情形并入终止情形，但有时仍必须单独处理。自定义证明策略有助于减少代码重复；不过，尽管本质上需要证明三个行为保持陈述，我们对循环展开的完整定义和证明也仅约 700 行 Rocq 代码（不包括 indep_spec）。尽管如此，关于发散的余归纳证明在技术上往往较为微妙，也难以正确构造。

相比之下，小步语义能够自然地处理发散，并支持统一的基于模拟的论证，但代价是需要更复杂的逐步推理和更繁重的不变式管理，从而降低了模块化和抽象的可能性。具体而言，不变式（例如 $indep$ 循环外提中的不变式）往往必须贯穿步关系，并在推理过程中即时证明，这使得将小步证明拆分为不同且可复用的部分更加困难。

我们的行为框架要求语义具有确定性，才能实现完全的行为等价。我们接受这一要求，因为 CompCert 后端的大多数中间语言都是确定性的，包括 `Cminor` 和 `RTL`。当然，我们的方法并不局限于 `Cminor` ——它的主要缺点是缺乏对 goto语句的支持。更合适的候选方案可能是 `RTL`，它并非归纳定义，而是由控制流图构成。其大步语义可以抽象函数调用，从而支持对内联等过程间优化进行推理。还可以引入大步循环构造，以支持循环优化。

通常，循环展开等循环变换会以系统化的方式改变控制流图：存在一个从转换后的控制流图到原始控制流图的满射映射，并且该映射保留边。Gourdin、Monniaux 等人 [15, 7, 6] 提出了一种方法，利用这一事实，在类似于 `RTL`的小步语义上验证此类变换。而不是

直接证明小步模拟，他们采用翻译验证 [16]：这是一种强大的技术，在变换过程中同时生成见证，之后可用这些见证验证变换的正确性。作为验证器，他们使用一个执行符号化小步执行的解释器，并提供提示来验证所需的匹配不变式。基于这一方法，他们得以验证若干新的复杂变换，包括循环无关代码移动（基于循环旋转），以及更通用的惰性代码移动策略。

尽管这些方法能为一大类变换提供强有力的结果，但它们比我们“原生”的大步方法更复杂。由于它们需要控制流图态射，一些变换（例如操作重排序、循环融合或循环外提）并不天然适用于其框架；在我们看来，这些变换更适合用大步证明。

参考文献

1 Andrew W. Appel 和 Sandrine Blazy。面向小步 Cminor 的分离逻辑。载于 Klaus Schneider 和 Jens Brandt 编，《高阶逻辑定理证明》，第 5–21 页，柏林、海德堡，2007年。施普林格·柏林·海德堡。 2 Yves Bertot、Benjamin Grégoire 和 Xavier Leroy。基于数据流分析证明编译器优化的结构化方法。载于《证明与程序类型国际研讨会（TYPES 2004）修订精选论文集》，《证明与程序类型国际研讨会（TYPES 2004）修订精选论文集》第 3839 卷，第 66–81 页，法国茹伊昂若萨，2004年12月。施普林格。URL：https://inria.hal.science/inria-00289549, doi:10.1007/11617990。 3 Arthur Charguéraud。精化大步语义。载于 Matthias Felleisen 和 Philippa Gardner 编，《编程语言与系统》，第 41–60 页，柏林、海德堡，2013年。施普林格·柏林·海德堡。 4 Patrick Cousot 和 Radhia Cousot。归纳定义、语义与抽象解释。载于第19届 ACM SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL ’92，第 83–94 页，美国纽约州纽约市，1992年。计算机协会。 doi:10.1145/143165.143184。 5 Nils Anders Danielsson 和 Thorsten Altenkirch。混合归纳与余归纳。草稿，2009年。URL：https://www.cse.chalmers.se/~nad/publications/danielsson-altenkirch-mixing.pdf。 6 Léo Gourdin。形式化验证的编译器中的惰性代码变换。载于第18届 ACM 面向面向对象语言、程序与系统的实现、编译和优化国际研讨会论文集（ICOOOLPS 2023），第 3–14 页，美国纽约州纽约市，2023年。计算机协会。 doi:10.1145/3605158.3605848。 7 Léo Gourdin、Benjamin Bonneau、Sylvain Boulmé、David Monniaux 和 Alexandre Bérard。利用块模拟对优化进行形式化验证。《ACM 编程语言会议录》，7(OOPSLA2)，2023年10月。 doi:10.1145/3622799。 8 Daniel Kästner、Xavier Leroy、Sandrine Blazy、Bernhard Schommer、Michael Schmidt 和 Christian Ferdinand。弥合差距——形式化验证的优化编译器 CompCert。载于《系统安全工程进展：第25届安全关键系统研讨会论文集》，《系统安全工程进展：第25届安全关键系统研讨会论文集》，第 163–180 页，英国布里斯托尔，2017年2月。CreateSpace。 9 David Knothe。CompCert-loop。 https://github.com/knothed/CompCert-loop，2026年。补充材料。 10 Xavier Leroy。真实编译器的形式化验证。《ACM通讯》，52(7)：107–115，2009年7月。 doi:10.1145/1538788.1538814。 11 Xavier Leroy。形式化验证的编译器后端。《自动推理杂志》，43(4)：363–446，2009年12月。 doi:10.1007/s10817-009-9155-4。

12 Xavier Leroy。形式化验证编译器：究竟意味着什么？载于 Ioannis Chatzigiannakis、Michael Mitzenmacher、Yuval Rabani 和 Davide Sangiorgi 编，第43届自动机、语言与编程国际研讨会（ICALP 2016），《莱布尼茨国际信息学论文集》（LIPIcs）第 55 卷，第 2:1–2:1 页，德国达格斯图尔，2016年。达格斯图尔城堡—莱布尼茨信息学中心。URL：https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2016.2，doi:10.4230/LIPIcs.ICALP.2016.2。 13 Xavier Leroy。CompCert——研究目标。https://compcert.org/research.html，2023年。访问日期：2026-02-17。 14 Xavier Leroy 和 Hervé Grall。余归纳大步操作语义。《信息与计算》，207(2)：284–304，2009年。doi:10.1016/j.ic.2007.12.004。 15 David Monniaux 和 Cyril Six。形式化验证的循环不变代码外提及其他优化。《ACM嵌入式计算系统汇刊》，22(1)，2022年12月。 doi:10.1145/3529507。 16 A. Pnueli、M. Siegel 和 E. Singerman。翻译验证。载于 Bernhard Steffen 编，《系统构造与分析工具和算法》，第 151–166 页，柏林、海德堡，1998年。施普林格·柏林·海德堡。 17 Youngju Song、Minki Cho、Dongjoo Kim、Yonghyun Kim、Jeehoon Kang 和 Chung-Kil Hur。CompCertM：支持 C汇编链接和轻量级模块化验证的 CompCert。《ACM 编程语言会议录》，4(POPL)，2019年12月。 doi:10.1145/3371091。 18 Gordon Stewart、Lennart Beringer、Santiago Cuellar 和 Andrew W. Appel。组合式CompCert。载于第42届 ACM SIGPLAN-SIGACT 编程语言原理研讨会年会论文集，POPL '15，第 275–287 页，美国纽约州纽约市，2015年。计算机协会。 doi:10.1145/2676726.2676985。 19 Jaroslav Ševčík、Viktor Vafeiadis、Francesco Zappa Nardelli、Suresh Jagannathan 和 Peter Sewell。CompCertTSO：面向宽松内存并发的经验证编译器。《ACM杂志》，60(3)，2013年6月。 doi:10.1145/2487241.2487248。