arXiv:2601.13727v1 [cs.PL] 20 Jan 2026

VeriFast 基础：通过提示镜像对验证工具结果进行实用认证

Bart Jacobs[0000−0002−3605−249X]

鲁汶大学，计算机科学系，DistriNet 研究组bart.jacobs@kuleuven.be
https://distrinet.cs.kuleuven.be/people/BartJacobs

摘要。 VeriFast 是一种领先工具，用于模块化地形式化验证单线程和多线程 C 与 Rust 程序的正确性属性。它通过对每个函数单独进行符号执行来验证程序，利用用户以分离逻辑形式标注的前置条件、后置条件和循环不变式，并采用基于分离逻辑的内存符号表示。然而，该工具本身由约 3 万行 OCaml 代码编写，尚未经过形式化验证。因此，工具中的错误可能导致它错误地报告输入程序正确。本文报告了一项早期成果：扩展 VeriFast，使其在成功验证 Rust 程序后，生成一份 Rocq 证明脚本，以证明该程序符合以 Rocq 编码的 Rust 公理语义。这显著提升了 VeriFast 在安全关键领域的适用性。我们采用提示镜像：记录 VeriFast 符号执行过程中的关键信息，并用这些信息引导 Rocq 重放该过程。

关键词： 程序验证 · 分离逻辑 · Rust。

1 引言

本文结构如下：我们在 §1 回顾 VeriFast [7]，在 §2 介绍证明脚本的生成，并在 §3 讨论可靠性。最后，我们将在 §4 讨论局限性和相关工作，并给出结论。

2 背景：VeriFast

请看图 1 中的 Rust 程序。函数 `reverse` 接收指向一个由指针变量构成的链表头部的指针，就地反转该链表，并返回指向结果链表头部的指针（即原链表的最后一个节点）。该函数通过递归辅助函数 `reverse_iter` 实现：它反复从第一个给定链表的头部弹出一个节点，并将其推入第二个链表的头部，直到第一个链表为空，然后返回第二个链表。main 函数分配

```
/*@
pred llist(head: *mut u8) =
   if head == 0 {
      true
   } else {
      *(head as *mut *mut u8) |-> ?next &*& llist(next)
   };
@*/

unsafe fn reverse_iter(original: *mut u8, reversed: *mut u8) -> *mut u8
//@ req llist(original) &*& llist(reversed);
//@ ens llist(result);
{
   //@ open llist(original);
   if original.is_null() { return reversed; }
   let next = *(original as *mut *mut u8);
   *(original as *mut *mut u8) = reversed;
   //@ close llist(original);
   reverse_iter(next, original)
}

unsafe fn reverse(mut list: *mut u8) -> *mut u8
//@ req llist(list);
//@ ens llist(result);
{
   //@ close llist(0);
   reverse_iter(list, std::ptr::null_mut())
}

fn main()
//@ req true;
//@ ens true;
{
   unsafe {
      //@ close llist(0);
      let mut node1: *mut u8 = std::ptr::null_mut();
      //@ close llist(&node1 as *mut u8);
      let mut node2 = &raw mut node1 as *mut u8;
      //@ close llist(&node2 as *mut u8);
      let mut node3 = &raw mut node2 as *mut u8;
      //@ close llist(&node3 as *mut u8);
      let reversed = reverse(&raw mut node3 as *mut u8);
      reverse(reversed);
      std::process::abort();
   }
}
```

图 1. 一个经过 VeriFast 注释的 Rust 程序示例，用于反转由指针变量构成的链表

栈上有三个指针变量，将它们初始化为一个以 `node3` 为头节点的链表，对其调用 `reverse`，然后再次对得到的链表调用 `reverse`。

该程序使用 **`unsafe`** 块，因此 Rust 编译器无法保证不存在未定义行为。尤其是，它无法保证指针解引用是安全的。相反，我们使用 VeriFast[1] [7] 来验证不存在未定义行为。VeriFast 要求为每个函数添加前置条件和后置条件注释，并以分离逻辑的一种变体 [12,14]，来表达，使用递归分离逻辑谓词 `llist` 来断言拥有一个由指针变量构成的链表。指向断言 `*P |-> V` 断言拥有指针 `P`所指向的变量，且该变量当前存储的值为 `V`。分离合取 `A1 &*& A2` 断言 `A1` 和 `A2` 分别成立；合取项必须断言拥有相互不相交 的变量集合。在谓词定义中出现 `&*&`，表示该链表无环；在 `reverse_iter` 的前置条件中出现，则表示这两个链表没有重叠。指向断言的右侧可以是一个模 式；模式 `?X` 将所指向的值绑定到逻辑变量 `X`，其作用域延伸至外层断言的末尾。

为了验证程序，VeriFast 会分别对每个函数进行符号执行，并将内存表示为由所谓的堆块组成的列表[2]。堆块要么是指向块，要么是谓词块。指向块的形式要么是 points_to_(p, s)，表示拥有由指针 p 指向的一个可能尚未初始化的变量，其初始化状态为 s；要么是 points_to(p, v)，表示拥有位于 p 的一个已初始化变量，其值为 v。这里， p、 s和 v 是一阶逻辑项，其中可能包含符号（因此称为符号执行）。声明一个取了地址的局部变量会产生（即添加到符号堆中）一个 points_to_块；对取了地址的局部变量赋值时，会消耗（即从符号堆中移除；若未找到则失败）一个适当的 points_to_ 块，并产生一个 points_to 块。调用函数时，会消耗被调用函数的前置条件，然后产生其后置条件。消耗断言 `llist(list)` 时，若符号堆中不存在 a `llist`(ℓ) 块，则会失败；其中 ℓ 是一个项，且可证明它等于变量 `list` 的值。为了引入此类谓词块，用户通常必须在程序中插入**`close`** `p(args);` 幽灵命令，这些命令会消耗谓词 `p` 的主体，并产生一个谓词块 `p`(args)。此过程可以使用 **`open`** `p(args);` 幽灵命令反向执行。 `main` 和`reverse_iter` 的符号执行轨迹如图 2 所示。

[1] https://github.com/verifast/verifast [2] 这个名称有些名不副实。更恰当的名称应该是“资源块”，因为堆块也用于表示栈分配内存和其他资源。



图 2. 函数 `main` 和 `reverse_iter`（原始 ≠nullptr 情况）的符号执行轨迹，展示每个位置的符号堆、符号存储和路径条件。由于路径条件只会增加，而且在这些示例中符号存储也只会增加，因此我们只展示新增元素。生成一个堆块会将其添加到符号堆的最前端。新生成的堆块以绿色显示；接下来将被消耗的堆块用红色下划线标示。ConsumeChunk 提示以橙色显示（见 §3）。

3 证明脚本生成

VeriFast 现在可以选择在成功验证 Rust 程序后，为用于开发机器检查数学证明的 Rocq 系统生成证明脚本[3]，其中定义了常量 `bodies`（程序中所有函数的列表，其函数体以 Rust 的中级中间表示（MIR）的一个轻微变体[4] 表示）、 `preds`（所有 VeriFast 谓词定义的列表）、 `specs`（包含每个函数的前置条件和后置条件的列表）以及 `symex_trees`（对于每个函数，包含一个符号执行树的列表，其中含有供 Rocq 编码的符号执行算法使用的提示），并定义了一个单一目标 `bodies_are_correctpreds specs symex_trees bodies`，用固定证明表达每个函数相对于其规范的正确性。至少对于图 1 中的示例程序，该证明可以通过，这意味着在假设 `bodies` 忠实地表示该程序、 `specs` 忠实地表达所需的正确性属性，并且 `bodies_are_correct`是可靠的情况下（下一节将对此进行讨论），该程序确实是正确的。

`bodies_are_correct`[5] 本质上是用 Rocq 对 VeriFast 的符号执行算法进行的一个（目前还很不完整的）重新实现。主要区别如下：

– VeriFast 有时会执行自动化步骤。例如，在尝试消耗一个 points_to_(ℓ,_) 块时，如果找不到匹配的块，但找到了一个 points_to(ℓ, v) 块，就会自动打开后者，以获得 points_to_- (ℓ,some(v))。为简化 `bodies_are_correct` 的实现，这些自动步骤在 `symex_trees` 中被显式指定。– 尝试消耗一个块时，例如 points_to(ℓ,_)，如果 VeriFast 在符号堆中找到一个 points_to(ℓ', v) 块，就会询问 SMT 求解器是否$\ell = \ell'$。只有求解器返回 Valid 时，VeriFast 才会认为该块匹配。相反， `bodies_are_correct` 会将应当匹配的块的索引 k 作为 ConsumeChunk(k) 提示接收于 `symex_trees` 中。– VeriFast 使用 SMT 求解器判断路径条件是否可满足；如果不可满足，就会放弃该符号执行路径。相反，当路径应当不可行时， `bodies_are_correct` 会在 `symex_trees` 中接收 Done 提示。

`bodies_are_correct`（至少在系统当前状态下）会简化为一个简单的证明目标，旨在由 `repeat (split || intro || congruence || tauto)`所概括的一个简单固定策略脚本来证明。

[3] 图 1 示例生成的证明脚本可在 https://github.com/verifast/verifast/blob/master/tests/rust/purely_unsafe/ trivialest_反转_expected.v 查看。[4] 参见 https://github.com/verifast/verifast/blob/master/bin/rust/rocq/VfMir.v [5] 其定义位于 https://github.com/verifast/verifast/blob/ master/bin/rust/rocq/ SymbolicExecution.v；该文件引用了 https://github.com/verifast/verifast/blob/ master/bin/rust/rocq/Values.v，其中定义了类型 `Value`。

4 可靠性

我们已在 Rocq 中证明了[6], VF MIR 的一种公理语义[7] （AxSem）相对于 `bodies_are_correct`（符号执行，即 SymEx）的可靠性。该公理语义是在 Iris[9], 中表达的；Iris 是 Rocq 中用于分离逻辑的一个框架。此证明为符号执行使用 VeriFast 谓词和函数规约，以及根据局部变量的地址是否被获取而对其采用不同处理方式提供了依据；这些内容在 AxSem 中均未出现。可靠性证明基于一个模型[8] ，该模型用逻辑堆来表示 VeriFast 符号堆；逻辑堆本质上是指向_块的多重集_ （目前如此）。VeriFast 谓词和断言的含义是在该模型中归纳定义的；我们还证明了生成和使用断言，以及打开和关闭谓词的可靠性。

目前，公理语义本身尚未经过任何验证。它存在一个已知的不可靠之处：与 VeriFast 一样，目前它忽略 StorageLive 和 StorageDead MIR 语句，并将所有局部变量都视为在整个函数体中始终存活。修复这一问题后，我们希望验证 AxSem 相对于一种或多种其他语义的可靠性，例如 RefinedRust 所使用的 Radium 语义 [6], ，或未来 MiniRust 的 Rocq 编码 [10]。

5 结论

本文介绍了迈向 Rust 基础性 VeriFast 的第一步：通过提示镜像这一务实方法实现该目标；这一方法也很可能适用于其他验证工具。然而，还有许多工作有待完成，包括支持更多 Rust 特性（如循环、结构体、泛型和栈展开）以及更多 VeriFast 特性（如归纳数据类型、不动点函数、引理、分数权限，以及按照 RustBelt 验证安全抽象的语义良构性 [8]），并验证我们的公理语义。尽管当前实现十分精简，我们在设计时着眼于最终支持 VeriFast 的全部功能，包括未来与 Rust 指针别名规则和宽松原子操作相关的 VeriFast 扩展。

据我们所知，Rust 目前唯一已有的基础验证工具是 RefinedRust [6]。其他语言的基础验证工具包括用于 C 语言的 VST[1,17] 和 RefinedC [15] 。然而，这些工具都直接在 Rocq 中实现，这限制了用户体验，也使得它们无法使用 SMT 求解器进行高性能自动化。

提示镜像是证书生成（例如 [3,2]）和预言机引导的结果验证（例如 [11]）的一种特殊情况。不过，尽管已有研究致力于为非基础性的程序验证工具提供基础性支撑（例如 Boogie [13],Why3 [4]和 Viper [5]），但据我们所知，目前还没有将提示镜像应用于基于分离逻辑的程序验证工具的工作。

[6] 参见 https://github.com/verifast/verifast/blob/master/bin/rust/rocq/SymexSoundness。v
[7] 参见 https://github.com/verifast/verifast/blob/master/bin/rust/rocq/AxSem.v
[8] 参见 https://github.com/verifast/verifast/blob/master/bin/rust/rocq/LogicalHeaps.v

利益披露。无。

参考文献

1. Appel, A.W.：经验证的软件工具链。载于：第20届欧洲编程语言与系统会议论文集：欧洲软件理论与实践联合会议的一部分。第1–17页。ESOP'11/ETAPS'11, Springer-Verlag，柏林，海德堡（2011） 2. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.：通过证明见证将 SAT/SMT 求解器模块化集成到 Coq 中。载于：Jouannaud, J.P., Shao, Z.（编）《认证程序与证明》。第135–150页。Springer Berlin Heidelberg，柏林，海德堡（2011） 3. Blanchette, J.C., Böhme, S., Paulson, L.C.：扩展 Sledgehammer 以支持 SMT 求解器。载于：Bjørner, N., Sofronie-Stokkermans, V.（编）《自动演绎—CADE-23》。第116–130页。Springer Berlin Heidelberg，柏林，海德堡（2011） 4. Cohen, J.M., Johnson-Freyd, P.：在 Coq 中形式化 Why3 核心。ACM 编程语言论文集 **8**(POPL)（2024年1月）。https://doi.org/10.1145/3632902, https://doi.org/10 3632902 5 1145. / . Dardinier, T., Sammler, M., Parthasarathy, G., Summers, A.J., Müller, P.：翻译式分离逻辑验证器的形式化基础。ACM 编程语言论文集 **9**(POPL)（2025年1月）。https://doi.org/10.1145/3704856, https://doi.org/10. 1145 3704856 6 / . Gäher, L., Sammler, M., Jung, R., Krebbers, R., Dreyer, D.：RefinedRust：一种用于高可信度验证 Rust 程序的类型系统。ACM 编程语言论文集 **8**(PLDI)（2024年6月）。https://doi.org/10.1145/3656422, https://doi.org/10. 1145 3656422 7 / . Jacobs, B., Vogels, F., Piessens, F.：Featherweight VeriFast。《计算机科学中的逻辑方法》**第11卷，第3期**，19（2015年9月）。https://doi.org/10.2168/ LMCS-11 1595 8 3 19 2015, (:) https://lmcs.episciences.org/ . Jung, R., Jourdan, J.H., Krebbers, R., Dreyer, D.：RustBelt：为 Rust 编程语言奠定安全的基础。ACM 编程语言论文集 **2**(POPL)（2017年12月 3158154 9 10 1145 3158154, 10 1145 ）。https://doi.org/ . / https://doi.org/ . / . Jung, R., Krebbers, R., Jourdan, J.H., Bizjak, A., Birkedal, L., Dreyer, D.：从头构建 Iris：高阶并发分离逻辑的模块化基础。《函数式编程杂志》**28**，e20（2018）。https://doi.org/10.1017/S0956796818000151 10. Jung, R. 等：MiniRust（2025），https://github.com/minirust/minirust 11. Leroy, X.：形式化验证的编译器后端。《自动推理杂志》**43**(4)，363–446（2009），http://xavierleroy.org/publi/compcert-backend.pdf 12. O'Hearn, P.W., Reynolds, J.C., Yang, H.：关于修改数据结构的程序的局部推理。载于：第15届计算机科学逻辑国际研讨会论文集。第1–19页。CSL '01，Springer-Verlag，柏林，海德堡（2001） 13. Parthasarathy, G., Müller, P., Summers, A.J.：对实用验证条件生成器进行形式化验证。载于：《计算机辅助验证：第33届国际会议，CAV 2021，线上活动，2021年7月20日至23日，论文集，第二部分》。第704–727页。Springer-Verlag，柏林，海德堡（2021）。https://doi.org/10.1007/ 978 3 030 81688 9_33 10 1007 978 3 030 81688 9_33 - - - - ,

https://doi.org/ . / - - - - . Reynolds, J.：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 计算机科学逻辑年度研讨会论文集。第55–74页（2002）。https://doi.org/10.1109/LICS.2002.1029817

15. Sammler, M.、Lepigre, R.、Krebbers, R.、Memarian, K.、Dreyer, D.、Garg, D.：《RefinedC：使用精化所有权类型自动化 C 代码的基础验证》。载于：第 42 届 ACM SIGPLAN 编程语言设计与实现国际会议论文集，第 158–174 页。PLDI 2021，计算机协会，美国纽约州纽约（2021）。https: //doi.org/10 3454036 16 1145 3453483 3454036，10 1145 3453483 2024 . / . https://doi.org/ . / . . Coq 开发团队：《Coq 证明助手》（9 月）。https://doi.org/ 10 14542673 17 5281 14542673，10 5281 . /zenodo. https://doi.org/ . / zenodo. . Zhou, L.、Qin, J.、Wang, Q.、Appel, A.W.、Cao, Q.：《VST-A：一种具有基础可靠性的注释验证器》。Proc. ACM 编程语言。8(POPL)（2024 年 1 月）。https: // doi.org/10.1145/3632911，https://doi.org/10.1145/3632911