arXiv:2303.03972v1 [cs.PL] 7 Mar 2023

借助 hacc 实现编舞的认证编译

Luís Cruz-Filipe[0000−0002−7866−7484]、Lovro Lugović[0000−0001−9684−9567] 和 Fabrizio Montesi[0000−0003−4666−901X]

南丹麦大学数学与计算机科学系{lcf, lugovic, fmontesi}@imada.sdu.dk

摘要。对通信进程进行编程颇具挑战，因为这需要编写彼此独立的程序，使其在执行过程中适时执行兼容的发送和接收操作。将这项任务交给程序员，很容易导致错误。舞蹈编程通过为开发者提供高层抽象，让他们能够从全局视角描述所需的通信结构，从而应对这一挑战。给定一个编排，可以通过端点投影（*EPP*）自动生成其中各个进程的实现。尽管舞蹈编程能够避免通信实现中的人为错误，但舞蹈编程框架的正确性在很大程度上取决于其复杂编译器的正确性，因此，人们开始在定理证明器中对舞蹈编程理论进行形式化。本文以其中一种形式化工作为基础，构建了一套能够从编排生成可执行代码的工具链。

关键词： 舞蹈编程 · 经认证的编译 · Jolie · 形式化验证

1 引言

在传统的分布式编程中，程序员需要分别为每个进程（端点）编写实现，并负责确保不同程序中的发送和接收操作正确匹配。这种方法既繁琐，又容易出错 [7]。

在舞蹈编程 [8], 中，开发者从全局视角指定进程之间所需的通信。给定一个舞蹈编程程序（称为编排），可以通过称为端点投影（*EPP*） [9]的过程，自动生成所有相关进程的正确实现。这样可以避免通信操作编程中的人为错误，并带来重要的理论优势，例如通过设计实现无死锁性——从编排生成的分布式代码始终不会发生死锁，因为编舞语言没有用于表示不匹配通信的语法 [1]。除了这些正确性优势之外，这种方法还能节省时间，让程序员专注于所开发协议的整体设计。

定义并实现 EPP 在技术上颇为复杂 [13], , 因此人们开始使用交互式定理证明器来机械化舞蹈编程理论

形式 [3,4,6,12]。其中，核心编排（*CC*）的形式化及其到有状态过程（*SP*）过程演算的 EPP，旨在具有广泛的适用性 [5]。具体而言，其设计允许为通信添加任意元数据，并且不限定用于表达本地计算、数据、进程标识符等的语言。

在本工作中，我们首次利用 [5] 开发了 `hacc`（读作“hack”）：一个用于将 CC 中的编排编译为可执行代码的工具。对于该可执行代码的目标语言，我们使用面向服务的编程语言 Jolie [10]。此外，`hacc` 的设计充分考虑了未来的可扩展性，因此可以添加新的目标语言。

`hacc` 的架构由两个编译阶段组成（图 1）。第一阶段（投影）使用 EPP 将编排转换为以 SP 表示的过程程序抽象表示，并将其作为中间表示（IR）。该阶段经过认证：它使用从 Coq 形式化 [3]中提取的 Haskell 程序。第二阶段（代码生成）将 IR 中的抽象操作转换为可执行编程语言。目前，我们以 Jolie 为目标语言。该阶段未经形式化验证：由于这是一种同态变换，以直接明了的方式遵循 IR 的项结构，因此其正确性很容易通过人工检查直接确立。

我们的开发成果支持编写和执行 CC 编排 [2,3,4,9]。它还证实了这样一种非正式的主张：CC 的形式化在设计时考虑了此类应用所需的灵活性 [5]。特别是，我们应用于 Jolie 时无需对该形式化作任何修改，只需恰当地实例化其参数，并将其提取的数据类型与我们的 Haskell 编译流水线中的其他组件进行简单对接即可。

结构。 第 2 节概述了编译器的架构，并介绍了其实现。第 3 节说明了我们如何将进程语言映射到

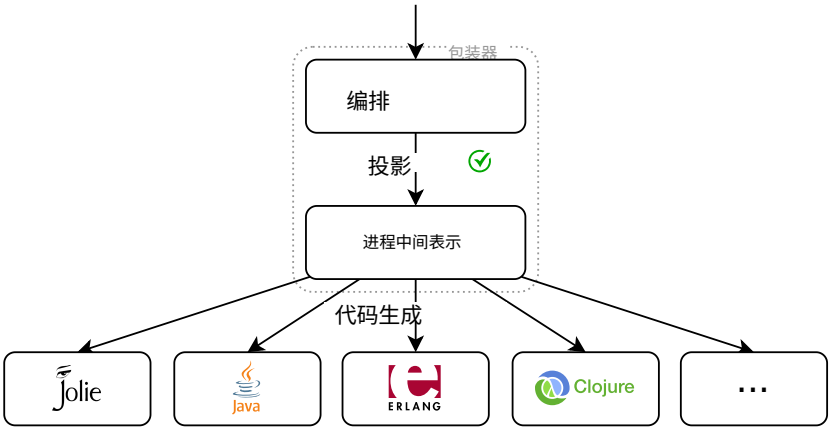

图 1. `hacc` 的编译流水线架构。

Jolie，以生成由多个独立服务组成的可执行 Jolie 代码。

相关工作。 目前还有另外两种对编舞程序的形式化。Kalas [12] 是一种在 Hol4 证明助手中形式化的编舞式编程语言。它配有一个端到端认证编译器，可将程序编译至 CakeML [11], ，这是 ML 语言中一个经过形式化验证的子集。相比之下，我们采用技术无关的方法，使我们的编译器基础设施能够复用于不同的目标语言。Pirouette [6]是一种在 Coq 中形式化的函数式编舞式编程语言，与 [4] 类似，它可以使用不同的本地计算语言进行实例化。不过，它尚未用于实现能够生成可执行代码的编舞式编译器。

2 hacc中的编排

我们使用 Haskell 数据类型表示编排和进程（分别是 CC 和 SP 的项）；这些类型是从 Coq 形式化中自动提取的。不过，鉴于 Coq 是一种依赖类型语言，而它能够提取到的语言都不是依赖类型语言，因此需要考虑一些问题。如前所述，该形式化对用于表示进程标识符、（递归）变量、本地计算语言中的项等的类型采取参数化处理。由于这是通过 Coq 的依赖类型特性实现的，提取出的 Haskell 代码有一些特殊之处。

具体而言，提取出的代码没有使用 Haskell 类型系统和参数多态，而是使用 Haskell 的 `Any` 类型来实现泛型。因此，要与提取出的代码交互，就必须使用 `unsafeCoerce`函数。为避免接口过于冗长，我们在提取出的代码外提供了一个更易用的薄包装器（图 1）。该包装器隐藏了必要的强制类型转换，并利用 Haskell 的参数多态对各项进行建模。这一步未经形式化验证，但包装器同样遵循 CC 和 SP 的项结构，因此手动检查其正确性并不困难。

图 2 展示了我们的接口。为简化起见，我们固定了进程标识符（`Pid`）、变量名（`Var`、`RecVar`）、选择标签（`Label`）和注释（`Ann`）的类型。 `Label` 类型是一个二元和类型，其构造器为 `CLeft` 和 `CRight` ；其他类型则是围绕 `String`构建、用作标识符的包装器。

编排具有类型 `Choreography e b`，其参数化依赖于本地计算语言和布尔表达式语言（`e` 和 `b`）。进程可以执行点对点交互（`Interaction`）——值通信（`Com`），即一个进程计算表达式并将结果发送给另一个进程；或选择（`Sel`），即一个进程通过传递标签来选择另一个进程应如何行为。交互包括注解（`Ann`），下文将对此进行讨论。条件语句（`CCond`）基于布尔表达式，而 `CCall` 调用一个具名过程。[1]

[1] CC 包含语义所需的运行时项。程序员不应显式编写这些项，因此我们不会将它们纳入数据类型。

```haskell
data Eta e = Com Pid e Pid Var | Sel Pid Pid Label

data Choreography e b
  = CEnd
  | Interaction (Eta e) Ann (Choreography e b)
  | CCond Pid b (Choreography e b) (Choreography e b)
  | CCall RecVar

newtype CDefSet e b = CDefSet [(RecVar, Choreography e b)]
newtype CProgram e b = CProgram (CDefSet e b, Choreography e b)
```

图 2. hacc 中编排的数据类型。

CProgram 是编舞程序的类型，其中包括具名过程的定义（CDefSet）以及主编舞。[2]

我们不直接使用数据类型，而是定义一组便捷的组合子来构建编排。我们提供一个 prog 组合子，它接收一对递归过程定义和主编舞，并据此返回一个 CProgram 。编排以指令列表的形式给出（通信、 CLeft 或 CRight 的选择、条件语句和调用），所有指令均由相应的组合子构建（分别为 com、 left、 right、 cond 和 call），再将这些组合子串联起来，生成我们表示中的一个 CC 项。

编译为 Jolie 时，程序员可以使用 ann 组合子为交互添加注释，从而配置生成的代码。这些注释将用于覆盖实现这些交互的 Jolie 操作的默认名称。

示例 *1*（分布式身份验证）。图 3 展示了 [3] 中的分布式身份验证编排，该编排编码为一个 CProgram。

[2] 在 Coq 中， DefSet 还包含参与每个过程的进程集合。理论上，这个集合可能无法计算，因为过程的数量可能是无限的。但手写的编排不会出现这种情况，因此我们的包装器会计算这个集合。

```haskell
auth :: CProgram String String
auth = prog ([],
  [ann "authenticate" $ com c "credentials" ip credentials,
   cond ip "check(credentials)"
     ([ann "authOk" $ left ip s, ann "authOk" $ left ip c,
       ann "acceptToken" $ com s "makeToken" c token],
      [ann "authFail" $ right ip s, ann "authFail" $ right ip c])])
  where [ip, s, c] = pids ["Ip", "Server", "Client"]
        [credentials, token] = vars ["credentials", "token"]
```

图 3。 CC 中的分布式身份验证编排。

此处， Client 希望向 Ip （身份提供方）进行身份验证，以便从 Server接收令牌。 Client 将其凭据发送给 Ip，后者会检查凭据，并将结果告知 Client 和 Server。如果身份验证成功， Server 会向 Client发送令牌，否则协议将终止。

本地计算语言和布尔表达式语言的项都是简单的字符串，会原样包含在生成的可执行代码中。◁

3 编译到 Jolie

进程的投影行为类型为 Behaviour e b，同样以本地语言为参数（图 4）。 Send 和 Recv 分别对应

Haskell

```haskell
data Behaviour e b
  = BEnd
  | Send Pid e Ann (Behaviour e b)
  | Recv Pid Var Ann (Behaviour e b)
  | Choose Pid Label Ann (Behaviour e b)
  | Offer Pid (Maybe (Ann, Behaviour e b)) (Maybe (Ann, Behaviour e b))
  | BCond b (Behaviour e b) (Behaviour e b)
  | BCall (RecVar, Pid)

newtype BDefSet e b = BDefSet [((RecVar, Pid), Behaviour e b)]
newtype Network e b = Network [(Pid, Behaviour e b)]
newtype BProgram e b = BProgram (BDefSet e b, Network e b)

epp :: CProgram e b -> Maybe (BProgram e b)
```

图 4. hacc 中进程的数据类型。

通信操作，而 BCond 和 BCall 与 CC 中的相同。其余术语 Choose 和 Offer 用于实现选择，即一个进程可以选择另一个进程应执行的所提供的行为。请注意，进程不必为两个标签都提供行为。

进程由标识符命名，并归入一个 Network，然后将其与每个进程的递归过程投影（BDefSet）配对，以形成一个 BProgram。最后， epp 使用提取出的 EPP 作为基础，将一个 CProgram 的投影应用于一个 BProgram。我们使用 Maybe 类型来处理编排不可投影的情况。

示例 2. 将示例 1 中的分布式身份验证编排进行投影，可得到图 5 所示的进程中间表示。为简洁起见，我们只展示 Client 的代表性情况。请注意，编排中的注释已保留并传播到投影中。◁

```haskell
BProgram (BDefSet [], Network [...,
  (Pid "Client",
   Send (Pid "Ip") "credentials" (Ann "authenticate")
     (Offer (Pid "Ip")
       (Just (Ann "authOk",
              Recv (Pid "Server") (Var "token") (Ann "acceptToken") BEnd))
       (Just (Ann "authFail", BEnd))))])
```

图 5. 示例 1 中编排的投影。

示例 *3.* 现在可以将示例 2 中的进程中间表示编译为可执行代码（图 6）。对于每个进程，后端都会生成相应的`service` Jolie 块。我们省略必要的部署配置，仅展示其行为，其结构遵循进程中间表示的结构。

在此示例中，注释用于指定服务公开的操作名称（`authOk` 和 `acceptToken` 用于 `Client`，`authenticate` 用于 `Ip`，等等）。一般而言，它们可以指定任意元数据，使程序员能够控制并引导后端的代码生成。◁

```jolie
service Client {
  ...
  main {
    authenticate@Ip( credentials )
    [ authOk() ] {
      acceptToken( token )
    }
    [ authFail() ] {
      nullProcess
    }
  }
}
```

图 6. 为示例 1 中的编排生成的可执行 Jolie 代码。

4 结论

我们实现了一套工具链，用于将舞蹈编程编译为 Jolie 中的可执行代码。复杂的步骤，即计算端点投影，由从 [3] 中的 Coq 形式化提取的认证代码处理。随后，这些认证代码与未经认证但正确性易于人工检查的包装器相结合。我们通过一个分布式身份验证协议演示了这套工具链。

参考文献

1. Carbone, M., Montesi, F.: 《设计即无死锁：多方异步全局编程》。载于：Giacobazzi, R., Cousot, R.（编）：POPL 论文集。第 263–274 2013 https://doi.org/10.1145/2429069.2429101 页。ACM（）。. Cruz-Filipe, L., 2 Montesi, F.: 《舞蹈编程的核心模型》。Theor. Comput. Sci. **802**，第 38–66 页（2020 年）。 https://doi.org/10.1016/j.tcs.2019.07.005, https://doi.org/10.1016/j.tcs.2019.07.005 3. Cruz-Filipe, L., Montesi, F., Peressotti, M.: 《认证舞蹈编译》。载于：Cerone, A., Ölveczky, P.C.（编）：计算理论方面—ICTAC 2021—第 18 届国际研讨会，虚拟会议，哈萨克斯坦努尔苏丹，2021 年 9 月 8–10 日，论文集。计算机科学讲义，第 12819 卷，第 115 133 2021 https://doi.org/10.1007/978-3-030-85315-0_8 – 页。Springer 4 （）。. Cruz-Filipe, L., Montesi, F., Peressotti, M.: 《在 Coq 中形式化图灵完备的舞蹈语言》。载于：Cohen, L., Kaliszyk, C.（编）：第 12 届交互式定理证明国际会议，ITP 2021，2021 年 6 月 29 日至 7 月 1 日，意大利罗马（虚拟会议）。信息学会议论文集，第 193 卷，第 15:1–15:18 页。达格斯图尔城堡—莱布尼茨信息学中心（ 2021 https://doi.org/10.4230/LIPIcs.ITP.2021.15 ）。. 5 Cruz-Filipe, L., Montesi, F., Peressotti, M.: 《舞蹈编程的形式理论》。CoRR **abs/2209.01886** （2022 年）。 https://doi.org/10.48550/arXiv.2209.01886 https://doi.org/10.48550/arXiv.2209.01886 , . Hirsch, A.K., Garg, D.: 《Pirouette：高 6 阶类型化函数式舞蹈》。ACM 编程语言论文集 **6**（POPL），第 1–27 页（2022 年）。 https://doi.org/10.1145/3498684 https://doi.org/10.1145/3498684 , . Leesatapornwongsa, 7 T., Lukman, J.F., Lu, S., Gunawi, H.S.: 《TaxDC：数据中心分布式系统中非确定性并发错误的分类》。载于：ASPLOS 论文集。第 517–530 页（2016 年）8. Montesi, F.: 《舞蹈编程》。博士学位论文，哥本哈根信息技术大学（2013 年），http://www.fabriziomontesi.com/files/choreographic- programming.pdf 9. Montesi, F.: 《舞蹈编程导论》。剑桥大学出版社（2023 年）10. Montesi, F., Guidi, C., Zavattaro, G.: 《使用 Jolie 进行面向服务编程》。载于：Bouguettaya, A., Sheng, Q.Z., Daniel, F.（编）：Web 服务基础，第 81–107 页。Springer（2014 年）。 https://doi.org/10.1007/978-1-4614-7518-7_4, https://doi.org/10.1007/978-1-4614-7518-7_4 11. Myreen, M.O., Owens, S.: 《从高阶逻辑生成证明的综合式 ML》。载于：函数式编程国际会议（ICFP）。第 115–126 页。ACM Press（2012 年 9 月）。 https://doi.org/10.1145/2364527.2364545, https://cakeml.org/icfp12/index.html 12. Pohjola, J.Å., Gómez-Londoño, A., Shaker, J., Norrish, M.: 《Kalas：面向舞蹈语言的经验证端到端编译器》。载于：Andronick, J., de Moura, L.（编）：第 13 届交互式定理证明国际会议，ITP 2022，2022 年 8 月 7–10 日，以色列海法。信息学会议论文集，第 237 卷，第 27:1–27:18 页。达格斯图尔城堡—莱布尼茨信息学中心（2022 年）。 https://doi.org/10.4230/LIPIcs.ITP.2022.27 https://doi.org/10.4230/LIPIcs.ITP.2022.27 , . Scalas, A., Yoshida, N.: 《少即是多：重新 13 审视多方会话类型》。ACM 编程语言论文集 **3**（POPL），第 30:1–30:29 页（2019 年）。 https://doi.org/10.1145/3290343, https://doi.org/10.1145/3290343